IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 2, 2019, accepted June 18, 2019, date of publication June 24, 2019, date of current version July 15, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2924550

Temporal Data Representation and Querying Based on RDF

FU ZHANG, KE WANG, ZHIYIN LI, AND JINGWEI CHENG
School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China

Corresponding author: Fu Zhang (zhangfu@cse.neu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61672139.

ABSTRACT With the explosive growth of temporal data, how to query and manage temporal data has become an important research issue. Resource description framework (RDF), as the standard data and knowledge description language of the semantic web, has been widely used to represent various domain data. Aiming at the representation and querying of temporal data, this paper proposes a temporal data representation model based on RDF and its corresponding querying method. First, a representation model called RDFt is proposed, which can represent temporal data with both the time information and the update count information, and the syntax and semantics of the RDFt model are given. Then, we propose a query language called SPARQL[t] for RDFt, and we give the query syntax and operations of SPARQL[t] in detail. In addition, a querying transformation algorithm from SPARQL[t] to SPARQL is proposed, in order to achieve compatibility with the existing RDF query engines. Finally, we implemented a prototype system that can support RDFt temporal data representation and querying, and the case studies and experimental results verify the feasibility of the proposed approach.

INDEX TERMS Temporal data, RDF, SPARQL, representation, querying.

I. INTRODUCTION

In recent years, with the development of the Semantic Web and knowledge engineering, Resource Description Framework (RDF) [1], which has formed a more systematic and comprehensive technical architecture in data and knowledge representation and processing, has become one of the main forms for representing knowledge. Based on the advantages of RDF in data and knowledge representation, many researchers have proposed to use RDF for temporal data representation and management in recent years. Using RDF data model to represent temporal data can ensure that the semantics of temporal data can be described accurately and flexibly, and also may help to realize the sharing of the temporal data in various applications. In addition, the efficient query mechanism of RDF can achieve effective querying on the temporal data.

In order to represent and manage temporal data in many practical fields (e.g., Geographic Information System GIS, sensor stream data, temporal database, dynamic social network, environmental meteorological monitoring system, and online financial data [2]), some temporal data representation models based on RDF have been proposed to describe the different granularity temporal data such as the time point and time interval. In general, there are three main types of temporal data models in the existing work, the first one is temporal data models based on the version control which is used to annotate the state of an RDF triple with the change of time; the second one is temporal data model based on the different extension forms of RDF triple (e.g., RDF quad-tuple syntax); the third one is temporal data model based on the original form of RDF triple by adding timestamp information after the predicate or the whole triple. Please refer to the related work in Section VI of this paper for more details. However, the existing representation models and query languages are not enough to represent and manage all types of temporal information in practical applications (as clearly listed in **Table 1**).

Furthermore, on the basis of the representation and querying requirements in Table 1, users may constantly put forward new requirements for queries. For example, a user may want to query the time of an event within a certain time interval, or the historical record of an event, or the state of the event at a time point. However, the existing models for representing

The associate editor coordinating the review of this manuscript and approving it for publication was Bora Onat.

TABLE 1. Some common requirements of temporal data representation and querying.

	Representation and querying requirements
R_1	Representing and querying time information about how many times an athlete has played for a team during a certain period of time
R_2	Representing and querying someone's career information over a certain period of time
R_3	Representing and querying a triple has several relevant historical records
…	…

temporal data cannot satisfy the requirements. The existing version-based snapshot model in [3] only can record the status information of each time period, and do not know how many records are related to this item. Also, the proposed multidimensional RDF model [4] is only suitable for the time flow information processing. The temporal RDF model [5], [6] can represent the temporal data, but cannot represent the update count information. Moreover, the temporal models in [7], [8], [9], [10] can represent only the state information that contains triples in a certain period of time. The model in [11] can represent only the N-dimensional time information. In addition, uncertain temporal model [12] and weighted-based temporal model [13] are proposed mainly for solving the problem of inconsistency in RDF knowledge bases with confidence annotations. Similarly, based on the fuzzy set theory, the temporal data model with annotation information in [14] can represent the fuzzy temporal information. The more detailed introduction and comparison can be found at the related work in Section VI of this paper.

Based on the observations above, in order to manage more variety of temporal information, in this paper we propose a new temporal data representation model called RDFt and its corresponding query language SPARQL[t]. The new model can represent temporal data with both the time information and the update count information that widely exist in the real-world applications. In general, **the paper makes the following main contributions:**

- We *propose a new temporal data representation model* RDFt in Section III, including: (*i*) we extend the RDF data model by adding time information and update count information to the predicate part of a triple, and we define the syntax and semantics of RDFt temporal data model; (*ii*) we provide several examples to well explain the model.
- We *propose a query language* called SPARQL[t] for RDFt in Section IV, including: (*i*) we define the basic query form and various query operations of SPARQL[t] in detail; (*ii*) we propose a complete querying transformation algorithm from SPARQL[t] to SPARQL in order to be compatible with the existing RDF query engines.
- We *implemented a prototype system* in Section V that can support RDFt temporal data representation and querying, including: (*i*) we introduce the architecture and some details of the prototype system; (*ii*) we test the prototype system based on two datasets of NBA basketball players and YAGO. The case studies and experimental results verify the feasibility of the proposed approach.

The remainder of this paper is organized as follows: Section II introduces some preliminaries. Section III proposes a new temporal data representation model RDFt, and gives the syntax and semantics. Section IV proposes a query language SPARQL[t] for RDFt, and proposes a complete querying transformation algorithm from SPARQL[t] to SPARQL. Section V implemented a prototype system and carried out some case studies based on two datasets of NBA basketball players and YAGO. Section VI introduces the related work. Section VII gives the conclusions and future work.

II. PRELIMINARIES

In this section we introduce some preliminaries on RDF, SPARQL, temporal data and Neo4J graph database.

A. RDF

Resource Description Framework (RDF [1]) and RDF Vocabulary Description Language (RDF Schema [15]) are the standard languages proposed by W3C (World Wide Web Consortium) to describe information resources on the Web. The basic structure of RDF is graph including nodes and edges. Two nodes and one edge consist of a triple, i.e., a triple includes "subject-predicate-object" three parts. Usually (s, p, o) is used to represent a basic triple, also known as an RDF statement. Each part in a triple is called resource and identified by a URI (Uniform Resource Identifier). Moreover, RDF allows blank nodes. The RDF Schema uses the notion of "class" to specify categories that can be used to classify resources. The relation between an instance and its class is stated through the "type" property. With RDF Schema one can create the hierarchies of classes by "sub-classes" and the hierarchies of properties by "sub-properties". Type restrictions on the subjects and objects of particular triples can be defined through "domain" and "range" restrictions. RDF has several syntaxes, such as: N-Triple, Turtle, JSON-based RDF Syntax, RDF/XML (XML Syntax for RDF), RDFa (for HTML and XML Embedding) [1], etc. Further, the semantics of an RDF model can be interpreted as shown in Definition 1 [1].

Definition 1 (Interpretation of an RDF Model): A simple interpretation I of an RDF model can be formally expressed as:

- A non-empty set IR of a resource, called the domain or universe of I;
- A subset of IP of IR, called the set of properties of I;
- A mapping $IEXT$ from IP into the powerset of $IR \times IR$, i.e. the set of sets of pairs $< x, y >$ with x and y in IR;
- A mapping IS from URI references in V into the union of IR and IP;
- A mapping IL is from typed literals in V into IR;

- A distinguished subset *LV* of *IR*, called the set of literal values, which contains all the plain literals in *V*.

The detailed introductions about RDF and RDF Schema can be found in [1] and [15].

B. SPARQL

SPARQL [16] (SPARQL Protocol and RDF Query Language) is a standard query language proposed by W3C for RDF data. It is a kind of SQL-like language and can help users query and modify data in datasets. Its basic query form is: select ?x [from dataset] where {*s p o.*}. The main purpose is to find the variable information that you want to query according to the pattern matching method and return it to the query user as a result. In SPARQL, any element of a triple (s, p, o) can be represented as a variable, and the first character of the variable is a question mark (?) or a dollar sign ($), which is commonly used as a question mark (?). Each RDF triple can be represented by an RDF graph. Multiple RDF graphs form an RDF group graph, i.e., a huge network graph structure. Therefore, the SPARQL language implements the query operation according to the pattern matching on the network graph structure.

- ASK: query whether there is a graph schema to be queried in RDF graph schema.
- SELECT: select the query result and return the binding of the query variable to the result.
- CONSTRUCT: create a new RDF graph.
- DESCRIBE: return a graph that contains all the information about the graph pattern matching nodes.

Moreover, SPARQL also supports filtering operations. According to the setting of filtering conditions, it can quickly find the data queried by users and filter out the expected result sets. In addition, SPARQL provides modifier keywords such as LIMIT, OFFSET, ORDER BY, and so on to facilitate users to find operation result sets. The detailed introductions about SPARQL can be found in [16].

C. TEMPORAL DATA

Temporal data is a data column recorded by the same index in chronological order. In general, temporal data is mainly divided into two types [7], [17]: one is transaction time, which indicates the time when a user accesses a database to perform data operations such as insertion, deletion and modification, that is, it is the time when a fact enters and stores in the database; the another one is valid time, which indicates that the current time information of the fact. On the basis of RDF, time information is added into the RDF model to represent temporal data, which can be divided into three main extension forms: the first one is temporal data model based on the version control which is used to annotate the state of an RDF triple with the change of time; the second one is temporal data model based on the different extension forms of RDF triple (e.g., RDF quad-tuple syntax), and the third form is temporal data model based on the original form of RDF triple by adding timestamp information. The current RDF model and its extensions are not good enough for representing all kinds of temporal data. Please refer to Section VI for more details.

D. Neo4J GRAPH DATABASE

In our prototype system we choose Neo4J graph database to store the temporal data. Neo4J [18], which is a high-performance NoSQL database, has been widely used to store the RDF data. Neo4J is easy to expand to hundreds of millions of levels of nodes and relationships, and it also provides traversal operation to retrieve data at high speed. The traversal operation is equivalent to the connection operation in relational database. Moreover, being similar to SQL and SPARQL, Cypher is a special declarative query language for querying Neo4J graph database. Cypher uses the expression method of graph pattern matching which is based on SPARQL. Please refer to [18] for more details about Neo4J graph database.

III. TEMPORAL DATA REPRESENTATION MODEL RDFt

In order to represent temporal data with both the time information and the update count information, in this section we propose a new temporal data representation model RDFt. We define the syntax (see part *A*) and the semantics (see part *B*) of the RDFt model in detail, and we also provide some running examples to well explain the model.

A. RDFt SYNTAX

For representing temporal data with both the time information and the update count information, our basis idea is to retain the RDF triple form and then extend the triple by adding the time information and update count information into the predicate p of the triple (s, p, o).

Formally, the sequential information t in RDFt data model refers to valid time, which can be either a time point or a time interval. The boundary value of the time interval is expressed by the start time point (ts) and the end time point (te). In addition, in order to further describe the update count information, an attribute tag n is added. **Figure 1** shows the representation syntax of the RDFt model:

Time point: $(s, p[t]\text{-}n, o), t \in T, n \in N$
Time interval: $(s, p[ts, te]\text{-}n, o), ts, te \in \mathrm{T}$ and $ts \leq te, n \in N$.

FIGURE 1. The syntax of the RDFt data model.

In the RDFt model:

- (s, p, o) is a standard RDF triple form "subject-predicate-object" as mentioned in Section II.
- $p[t]\text{-}n$ or $p[ts, te]\text{-}n$, which is the predicate part of a triple, represents the relationship between the subject s and the object o. In detail, [] denotes an optional item, the datatypes of t, ts and te are *xsd*: *date*, and $T = \{[ts, te] \mid xsd\text{: } date\}$ is the time domain. Note that, the time point information is represented as t, and the time interval

information is represented as $[ts, te]$. Furthermore, when $ts = te$, the time interval $(s, p[ts, te]\text{-}n, o)$ is equivalent to the time point $(s, p[t]\text{-}n, o)$, where $t = ts = te$.

- n in $p[t]\text{-}n$ or $p[ts, te]\text{-}n$ denotes the update count information, i.e., the triple records are updated n times. The default value of n is 1. The change of n is based on the change of transaction time, and the largest n represents the most recent historical record of the triple. Based on the update count n, we can quickly find out the change records of the triple by ranking the query result set. Particularly, when any number m in 1-n (i.e., $1 \leq m \leq n$) is not included in the query result set, it denotes that the triple with the update count m is deleted.

To well explain the model, **Figure 2** shows an example of the RDFt data model.

Example 1: representing the personal information of NBA player LeBron James with the RDFt data model.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdft: <http://www.neu.edu/2018/rdft-syntax-ns#> .
@prefix info: <http://www.neu.edu/2018/NBA_Sportinfo#> .

<LeBron_James> <Name> "LeBron_James" .
<LeBron_James> hasBirthCity[1984-12-30]-1 "Akron_OH" .
<LeBron_James> <Stature> "2.03"^^xsd:decimal .
<LeBron_James> <Plays_For[2003-06-06,2004-05-07]-1>
"Cleveland_Cavaliers" .
<LeBron_James> <Plays_For[2004-09-14,2005-04-26]-2>
"Cleveland_Cavaliers" .
<LeBron_James> <Plays_For[2005-04-18,2006-08-08]-3>
"Cleveland_Cavaliers" .
<LeBron_James> <Plays_For[2006-05-27,2007-11-09]-4>
"Cleveland_Cavaliers" .
<LeBron_James> <Plays_For[2010-01-03,2011-09-06]-1>
"Miami_Heat" .
<LeBron_James> <Plays_For[2011-01-11,2012-08-20]-2>
"Miami_Heat" .
<LeBron_James> <Score1[2003-06-06,2004-05-07]-1>
"1654"^^xsd:integer .
<LeBron_James> <Score1[2004-09-14,2005-04-26]-2>
"2175"^^xsd:integer .
<LeBron_James> <Score1[2005-04-18,2006-08-08]-3>
"2478"^^xsd:integer .
<LeBron_James> <Score1[2006-05-27,2007-11-09]-4>
"2132"^^xsd:integer .
<LeBron_James> <Score1[2007-04-26,2008-07-25]-5>
"2250"^^xsd:integer .
<LeBron_James> <Score1[2008-03-10,2009-04-10]-6>
```

FIGURE 2. An example of RDF t data model.

The example of RDFt data model in Figure 2 shows the personal information of NBA stars, including the information about James' regular season team, game time and score (time in the smallest unit of day). From Figure 2, the model can not only represent the temporal information, but also the update count information with the change of time.

Furthermore, we need to define the RDFt vocabulary based on the RDF vocabulary [1], [15] by adding a new RDFt namespace (@prefix rdft:<http://www.neu.edu/2018/rdft-syntax-ns>) and several RDFt properties *rdft*: *hasTime*, *rdft*: *hasStartTime*, *rdft*: *hasEndTime*, *rdft*: *hasNumUpdate*. As we have known, reification of an RDF statement is a mean of providing metadata for that statement, and it is the action that enables the statement to be used as the subject or object in another RDF statement [1]. On the basis, we first give the new RDFt vocabulary in **Table 2**.

TABLE 2. RDFt Specification vocabulary.

RDFt Reification Vocabulary	Type	Explanation
rdf:Statement	class	Representing a statement of a triple.
rdf:subject	property	Representing the subject of a statement.
rdf:predicate	property	Representing the predicate of a statement.
rdf:object	property	Representing the object of a statement.
rdft:hasTime	property	Representing a property with a time point.
rdft:hasStartTime	property	Representing a property that has a start time.
rdft:hasEndTime	property	Representing a property that has an end time.
rdft:hasNumUpdate	property	Representing the property of the update count of a triple record.

The reification of RDFt $(s, p[t]\text{-}n, o)$ is as follows:

```
?statement  rdf:type  rdf:Statement .
?statement  rdf:subject  s .
?statement  rdf:predicate  ?p .
?statement  rdf:object  o .
?p  a  rdf:Property .
?p  rdft:hasTime  t .
?p  rdft: hasNumUpdate  n .
```

FIGURE 3. The form of RDFt reification.

On the basis of Table 2, in the following we further give the form of RDFt reification as shown in **Figures 3 and 4**:

- if t is the time point, $t \in T$, $n \in N$.
- if t is the time interval, $t = [ts, te]$, $ts, te \in T$ and $ts \leq te$, $n \in N$.

Based on the RDFt vocabulary and reification form as mentioned above, we choose a part of James' personal information in Figure 2, and further give their reification forms as shown in **Figure 5**. In detail, among the triples in Figure 2, we choose two triples as shown in Figure 5, the first triple represents that LeBron_James was born in "Akron_OH" on "1984-12-30", and the second triple indicates that his

The reification of RDFt $(s, p[ts, te]\text{-}n, o)$ is as follows:

?statement rdf:type rdf:Statement .
?statement rdf:subject s .
?statement rdf:predicate ?p .
?statement rdf:object o .
?p rdf:type rdf:Property .
?p rdft:hasStartTime ts .
?p rdft:hasEndTime te .
?p rdft: hasNumUpdate n .

FIGURE 4. The form of RDFt reification.

third team is "Cleveland_Cavaliers" between "2005-04-18" and "2006-08-08". In Figure 5, *_:xxx* and *_:yyy* are anonymous empty nodes representing the aggregation forms of two triples.

For the following two triples about the James' personal and his career information in Figure 2.

<LeBron_James> <hasBirthCity[1984-12-30]-1> "Akron_OH" .
<LeBron_James> <Plays_For[2005-04-18,2006-08-08]-3> "Cleveland_Cavaliers" .

The reification forms of two triples are as follows:

_:xxx rdf:type rdf:Statement .
_:xxx rdf:subject <LeBron_James> .
_:xxx rdf:predicate ?hasBirthCity .
_:xxx rdf:object "Akron_OH" .
?hasBirthCity rdf:type rdf: Property.
?hasBirthCity rdft:hasTime "1984-12-30"^^xsd:date .
?hasBirthCity rdft:hasNumUpdate "1"^^xsd:integer .
_:yyy rdf:type rdf:Statement .
_:yyy rdf:subject <LeBron_James> .
_:yyy rdf:predicate ?Plays_For .
_:yyy rdf:object " Cleveland_Cavaliers " .
?Plays_For rdf:type rdf: Property .
?Plays_For rdft:hasStartTime "2005-04-18"^^xsd:date .
?Plays_For rdft:hasEndTime "2006-08-08"^^xsd:date .
?Plays_For rdft:hasNumUpdate "3"^^xsd:integer .

FIGURE 5. RDFt reification examples.

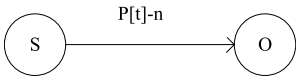


FIGURE 6. An RDFt graph.

Being similar to RDF, each RDFt triple can be represented as an RDFt graph as shown in **Figure 6**, and multiple RDFt triples form an RDFt graph. Here, S and O represent subject and object respectively, and $p[t]$-n is a predicate with the time and update count information.

Further, according to the time point and time interval information, the Figure 6 can be further represented by the following **Figures 7 and 8**.

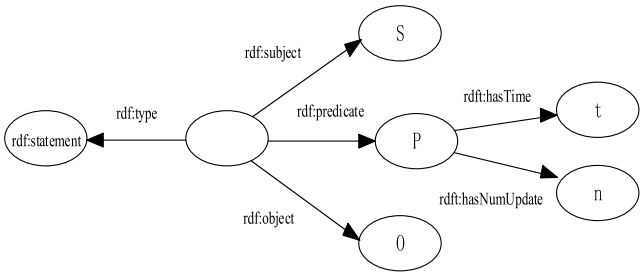


FIGURE 7. An RDFt graph with the time point information.

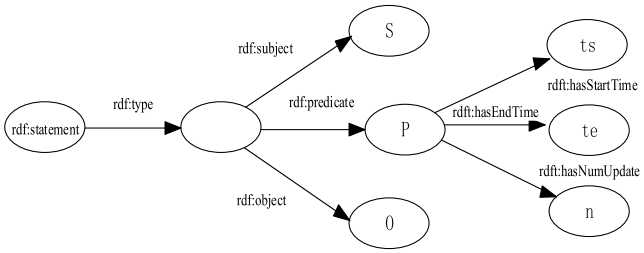


FIGURE 8. An RDFt graph with the time interval information.

- When t is a time point, an empty node can be introduced and the predicate part P can be further represented by the RDFt vocabulary in Figure 7.
- When t is a time interval, i.e., $t = [ts, te]$, an empty node can be introduced and the predicate part P can be further represented by the RDFt vocabulary in Figure 8.

From the Figures 7 and 8, an RDFt graph can be equivalently transformed into an RDF graph by introducing the RDFt vocabulary and the blank nodes.

B. RDFt SEMANTICS

The semantics of RDFt mainly includes three aspects: interpretation, satisfaction, and entailment. Interpretation means to use expressions or logical relational operators to give an explanation of the semantics of the data model; Satisfaction is the representation of the basic semantic relationship between an explanation and an expression; and Entailment expresses the logical relationship between two things, mainly used for knowledge reasoning and logical derivation.

On the basis of the RDF semantics in Definition 1 and the work in [5], [6], [17], [19], in the following we further give the semantics of the RDFt model.

Definition 2 (Temporal Interpretation): A temporal interpretation I of the temporal RDFt data model can be further defined by adding the following constraints into the Definition 1:

- A subset T of the resource set IR, representing the time point or interval information.
- A value NT means there is no time label information in a triple.
- The property set IP adds several temporal properties such as *rdft*: *hasTime*, *rdft*: *hasNumUpdate*, *rdft*: *hasStartTime*, and *rdft*: *hasEndTime*.

- A subset BP of IR, called basic properties, which are the predicate properties without time information.
- A mappingPT from $BP \times (T \cup \{NT\}) \times (T \cup \{NT\})$ to IP.

Further, giving a tuple $tp = PT(bp, ts, te, n)$, where tp represents a property with time information and bp represents a basic property, the temporal interpretation I and the mapping function PT can be further illustrated as follows:

- $IEXT$(rdf:property) contains properties $<tp, bp>$.
- If ts is not NT, then $IEXT(rdft{:}hasStratTime)$ contains $<tp, ts>$, otherwise, $IEXT(rdft{:}hasStartTime)$ does not contain any value pair $<tp, t>$ at the time t.
- Similarly, if te is not NT, then $IEXT(rdft{:}hasEndTime)$ contains $<tp, te>$, otherwise, $IEXT(rdft{:}hasEndTime)$ does not contain any value pair $<tp, t>$ at the time t.
- When $ts = te$, i.e., $t = ts = te$, then $IEXT(rdft{:}hasTime)$ contains $<tp, t>$.

Satisfaction refers to the basic semantic relationship between the interpretation I and RDFt triples. When the interpretation I satisfies the RDFt triples, which means that the RDFt triples meet all the conditions described in I, then the RDFt triples must be true. Since that not all of interpretations are reasonable or make sense in some given RDFt databases, and it is possible that they do not match the data that actually exists in the databases. Therefore, in the following we give the definitions of RDFt satisfaction and entailment.

Definition 3 (RDFt Satisfaction): Given an interpretation I, I satisfies a triple e in the RDFt model (written as $I \models e$) with the following constraints:

- $I \models (s, p[t]\text{-}n, o)$ iff $\forall t \in T, (s, p, o) \in I(t), n \in N$.
- $I \models (s, p[ts, te]\text{-}n, o)$ iff $\forall ts \in T, te \in T, (s, p, o) \in (I(ts) \wedge I(te)), n \in N$.
- $I \models (sp, rdfs{:}subPropertyOf, p)$ iff $\forall t \in T, (s, sp, o) \in I(t), (s, p, o) \in I(t)$.

Further, if $\forall e \in D$ and $I \models e$, then I satisfies the RDFt data model D (written as $I \models D$).

Definition 4 (RDFt Entailment): Given an interpretation I of an RDFt model, if $I \models (s, p[t]\text{-}n, o)$, then $I \models (s, sp[t]\text{-}n, o)$, where sp is a subproperty of p.

IV. TEMPORAL DATA QUERY LANGUAGE SPARQL[t]

Based on the proposed RDFt data model, in this section we further propose a temporal data query language SPARQL[t] by extending RDF standard query language SPARQL [16]. We first define the query syntax and operations of SPARQL[t] (see part *A*). Then, we propose a complete querying transformation algorithm from SPARQL[t] to SPARQL in order to achieve compatibility with the existing RDF query engines (see part *B*).

A. SPARQL[t] QUERY SYNTAX AND OPERATIONS

Being similar to the SPARQL query, the query form of SPARQL[t] includes the following parts: declaration, result set, dataset, graph pattern, and result modifier as shown in **Figure 9**.

Declaration: *PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ... .*
Result set: SELECT variable name
Dataset: FROM <Dataset>
Graph pattern: *WHERE { RDFt Graph patterns .*
FILTER (condition expressions) .}
Result modifier: ORDER BY, LIMIT, ...

FIGURE 9. SPARQL[t] query syntax.

where:

- Declaration: It is a unified statement of namespaces.
- Result set: SPARQL[t] uses the SELECT, ASK, DESCRIBE, and CONSTRUCT keywords to query. The result of pattern matching query is a result set or multiple RDFt graphs. Among them, SELECT is a standard query form and return all or part of the list variable through the pattern matching binding value; ASK returns the result yes/no, i.e., querying whether the data set contains the RDFt query graph or not; DESCRIBE returns all RDFt data related to a certain resource; CONSTRUCT generates an RDFt graph according to the result of the query graph, and returns the result as a graph.
- Dataset: It is the dataset to be queried and is identified by URL enclosed in angle brackets. The dataset can include named data sets, or a default graph with no names.
- Graph pattern: It includes basic graph pattern and complex graph pattern. The basic graph pattern is a collection of triple patterns, e.g., (?*p rdft:hasTime t*^^xsd:date) is a triple pattern, where the symbol ? is the prefix of the variable. In a basic graph pattern, the relationship of the triple patterns is intersection. The complex graph pattern includes group graph pattern, optional graph pattern, and multi-graph union pattern as will be introduced in detail in the part *B* of this section.
- Filter conditions: We can use the FILTER keywords to filter out the required information.
- Result modifier: It inherits the result modifier in SPARQL, and the following common modifiers are included: DISTINCT can ensure that the result of the query is unique in the result sequence; ORDER BY can sort the results; OFFSET can control the start position of the result returned in the result sequence; LIMIT can limit the number of results in the query result sequence.

In particular, in order to query the temporal information in our proposed RDFt model in section III, on the basis of the SPARQL query language, we need to further add several new syntax constructors as shown in **Table 3**.

Figure 10 provide an example of SPARQL[t] query, which is used to query the time and total scores when James played for the Cleveland Cavaliers at his sixth career as mentioned in Figure 2.

Moreover, the SPARQL[t] basic query operations can further manage RDFt datasets and include operations INSERT, DELETE, and UPDATE. The INSERT operation allows to insert triples into an RDFt dataset, the DELETE operation

TABLE 3. Several new SPARQL[t] query syntax constructors.

	QueryUnit	::=	Query
[1]	GroupGraph Pattern	::=	GraphPattern Union GraphPattern| GraphPattern GraphPattern| GraphPattern OPTIONAL GraphPattern
[2]	Graph Patterns	::=	Subject Predicate[*ts*, *te*]-*n* Object . |Subject Predicate[*t*]-*n* Object . |Subject Predicate Object .
[3]	*t*	::=	xsd:Date
[4]	*ts*	::=	xsd:Date
[5]	*te*	::=	xsd:Date
[6]	*n*	::=	xsd:Integer

Example 2: querying time and total scores when James played for the Cleveland Cavaliers at his sixth career as mentioned in Figure 2:

```
@base <http://yago-knowledge.org/resource/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdft:<http://www.neu.edu/2018/rdft-syntax-ns#> .
@prefix info:<http: // www.neu.edu /2018/NBA_Sportinfo# > .
SELECT ?ts ?te ?score
WHERE {
  ?SportName info:Plays_For[?ts,?te]-?n "Cleveland_Cavaliers" .
  ?SportName info:Score1[?ts, ?te]-?m ?score .
  FILTER ?SportName= "LeBron_James" and ?n=6 }
```

FIGURE 10. An example of SPARQL[t] query.

allows to delete triples from a dataset, and the UPDATE operation supports the INSERT and DELETE operations.

B. QUERY TRANSFORMATION FROM SPARQL[t] to SPARQL AND CYPHER

In order to achieve compatibility with the existing SPARQL query engines, in this section we further give the query transformation algorithm from SPARQL[t] to RDF standard query language SPARQL. Moreover, Neo4J [18] is a very popular graph database to store RDF datasets [20], and thus our implemented prototype as will be introduced in the subsequent section V also uses the Neo4J graph database to store the temporal data. Cypher is a special declarative query language for querying Neo4J graph database. Therefore, we also give the query transformation algorithm from SPARQL[t] to Cypher. In this section:

- we first give the transformation rules and algorithms from each part of SPARQL[t] *query forms* to the corresponding query forms of SPARQL and Cypher (see part *B.1*).
- Then, we give the transformation rules and algorithms of SPARQL[t] *basic graph pattern* to SPARQL and Cypher (see part *B.2*).
- Final, we further give the transformation rules and algorithms of SPARQL[t] *complex graph pattern* to SPARQL and Cypher (see part *B.3*).

1) TRANSFORMATION OF QUERY FORMS

From the previous part *A*, the SPARQL[t] query forms can be divided into the following several main parts:

(1) *Transformation of declaration in SPARQL[t]*

The declaration in SPARQL[t] is the same as the declaration in SPARQL. The namespace is declared at the beginning of a query, and the newly added namespace in SPARQL[t] is the prefix *rdft*. Further, when transforming from SAPRQL to Cypher query language, the namespace can be stored in the Neo4J database as a property of a resource.

(2) *Transformation of result set in SPARQL[t]*

- Transformation of SELECT

The SELECT in the result set of the query can be directly transformed from SPARQL[t] to SPARQL. Noted that, the result set of the query in Cypher is placed after the RETURN keyword, and the SELECT in SPARQL[t] and SPARQL can be directly transformed into the RETURN in Cypher.

- Transformation of ASK

The ASK in the result set of the query has the same function in both SPARQL[t] and SPARQL. But the Cypher query language does not support such query keyword, and thus we need to use MATCH function in Cypher to replace the function of ASK.

- Transformation of DESCRIBE

The DESCRIBE is to query all attributes related to a resource or a variable. The function is supported in both SPARQL[t] and SPARQL, and thus the transformation is direct. In Cypher, the search path can link all associated information to find all the attributes related to the resource.

- Transformation of CONSTRUCT

The CONSTRUCT in the result set of the query can be directly transformed from SPARQL[t] to SPARQL. But such function is not supported directly in Cypher, in order to replace the function, we need to use the CREATE keyword in Cypher to re-create the result graph.

(3) *Transformation of FROM in SPARQL[t]*

The FROM is to specify the dataset to be queried and is identified by the URL enclosed in angle brackets in both SPARQL[t] and SPARQL. Also, in Cypher the queried dataset is directly stored in the Neo4J database. Therefore, the transformation of FROM is direct in the three query languages.

(4) *Transformation of WHERE in SPARQL[t]*

In SPARQL[t], WHERE is followed by RDFt graph patterns. Therefore, we need to define some rules as will be shown in the following Table 4 to transform the RDFt graph patterns into RDF graph patterns in SPARQL and Cypher.

TABLE 4. The Transformation rules from SPAEQL[t] to SPARQL and Cypher.

	SPARQL[t]	SPARQL	Cypher

(1). Using the SELECT keyword to query when the time t is an instant. That is, when t = ts = te	Select ? object
	Where {
	subject predicate[t]-n object .
	}
		Select ? object
		Where {
		subject ?p object .
		?p rdf:type rdf:property .
		?p rdft:hasTime t^^xsd:date .
		?p rdft:hasNumUpdate n . }
			MATCH
			(subject)-[?p{type:property,
			hasTime:t^^xsd:date,
			hasNumUpdate:n}]
			→(object)
			RETURN object

(2). Using the SELECT keyword to query when the time t is an interval. That is, t = [ts, te], and ts < te	Select ? object
	Where {
	subject predicate[ts,te]-n object
	.
	}
		Select ? object
		Where {
		subject ?p object .
		?p rdf:type rdf:property .
		?p rdft:hasStartTime ts^^xsd:date .
		?p rdft:hasEndTime te^^xsd:date .
		?p rdft:hasNumUpdate n . }
			MATCH
			(subject)-[?p{type:property,
			hasStartTime:ts^^xsd:date,
			hasEndTime:te^^xsd:date,
			hasNumUpdate:n}]
			→(object)
			RETURN object

(3). When there is no time information on the property, i.e., the pattern is an ordinary triple	Select ? object
	Where {
	subject predicate object .
	}
		Select ? object
		Where {
		subject predicate object . }
			MATCH
			(subject)-[predicate]
			→(object)
			RETURN object

(4). The query statements with the UNION\ OPTIONAL\FILTER in SPARQL[t]	Select ? x ?y
	Where {
	{ RDFt Graph Pattern.
	} (OPTIONAL|UNION) {
	RDFt Graph Pattern.
	FILTER (condition1) }
	FILTER (condition2) }
		Select ? x ?y
		Where {
		{ RDF Graph Pattern.
		} (OPTIONAL|UNION)
		{ RDF Graph Pattern.
		FILTER (condition1) }
		FILTER (condition2) }
			MATCH
			(s)-[p{…}]-(o)
			(OPTIONAL|UNION)
			(s)-[p{…}]-(o)
			WHERE
			condition1 condition2
			RETURN s,o

(5). The ASK query in SPARQL[t]	ASK { subject predicate[t]-n object . }	ASK{ subject ?p object . ?p rdf:type rdf:property . ?p rdft:hasTime t^^xsd:date . ?p rdft:hasNumUpdate n . }	If the result of the match is the same as the result in MATCH, then YES is returned, otherwise NO. MATCH (subject)- [?p{type:property, hasTime:t^^xsd:date, hasNumUpdate:n}]→(object) RETURN subject,?p,object
(6). The DESCRIBE query in SPARQL[t]	DESCRIBE ?s Where {?s ?p[t]-n ?o .}	DESCRIBE ?s Where. {?s ?p o . ?p rdf:type rdf:property . ?p rdft:hasTime t^^xsd:date . ?p rdft:hasNumUpdate n . }	MATCH (s)-[?p{type:property, hasTime:t^^xsd:date, hasNumUpdate:n}]→(o) RETURN s
(7). The CONSTRUCT query in SPARQL[t]	CONSTRUCT {?s1 ?p1[t1]-n1 ?o1 .} Where {?s1 ?p2[t2]-n2 ?o2 . ?o1 ?p3[t3]-n3 ?o3 .}	CONSTRUCT {?s1 ?p1 o1 . ?p1 rdf:type rdf:property . ?p1 rdft:hasTime t1^^xsd:date . ?p1 rdft:hasNumUpdate n1 .} Where {?s1 ?p2 o2 . ?p2 rdf:type rdf:property . ?p2 rdft:hasTime t2^^xsd:date . ?p2 rdft:hasNumUpdate n2 . ?o1 ?p3 o3 . ?p3 rdf:type rdf:property . ?p3 rdft:hasTime t3^^xsd:date . ?p3 rdft:hasNumUpdate n3 . }	MATCH (o1)- [?p3{type:property, hasTime:t3^^xsd:date, hasNumUpdate:n3}]→(o3), (s1)- [?p2{type:property, hasTime:t2^^xsd:date, hasNumUpdate:n2}]→(o2) WITH (s1),(o1) CREATE (s1)- [?p1{type:property, hasTime:t1^^xsd:date, hasNumUpdate:n1}]→(o1)

(8). The result modifiers in SPARQL[t]: ORDER BY, LIMIT, OFFSET	Select ? object
	Where {
	subject predicate[t]-n object .
	}
	ORDER BY n
	LIMIT 5
	OFFSET 3
		Select ? object
		Where {
		subject ?p object .
		?p rdf:type rdf:property .
		?p rdft:hasTime t^^xsd:date .
		?p rdft:hasNumUpdate n .}
		ORDER BY n
		LIMIT 5
		OFFSET 3
			MATCH
			(subject)- [?p{type:property,
			hasTime:t^^xsd:date,
			hasNumUpdate:n}]→(object)
			RETURN object
			ORDER BY ?p.
			hasNumUpdate
			LIMIT 5
			SKIP 3

(9). The SPARQL[t] update operations, including INSERT and DELETE. When you want to modify a certain part of a triple, you can first DELETE the triple, then INSERT a new triple	DELETE DATA { s p[t]-n o .} INSERT DATA { s p[t]-n o .} DELETE { {s p[t]-n o .} } INSERT { {s p[t]-n o .} }	DELETE DATA { { s ?p o . ?p rdft:hasTime t . ?p rdft:hasNumUpdate n . }} INSERT DATA { { s ?p o . ?p rdft:hasTime t . ?p rdft:hasNumUpdate n . }} DELETE { { s ?p o . ?p rdft:hasTime t . ?p rdft:hasNumUpdate n . }} INSERT { { s ?p o . ?p rdft:hasTime t . ?p rdft:hasNumUpdate n . }}	MATCH (s)-[p]-(o) WHERE p.hasNumUpdate=3 REMOVE p.hasNumUpdate MATCH (s)-[p]-(o) WHERE id(s)=2 and id(o)=6 Set p.hasNumUpdate = 5 RETURN p

Also, we need to transform RDFt FILTER condition expressions into the RDF FILTER condition expressions. Moreover, the FILTER keyword is converted to the WHERE keyword in the Cypher language to indicate the constraints.

(5) *Transformation of result modifier in SPARQL[t]*

The main function of result modifiers is to sort or constrain the forms of the returned results, and thus the result modifiers in SPARQL[t] are the same as the result modifiers in SPARQL. Moreover, there are several minor differences of the result modifiers between SPARQL[t] and Cypher, and we need to establish the correspondences between them as will be shown in the following Table 4.

Based on the observations above, the following **Table 4** gives the detailed transformation rules from SPARQL[t] to SPARQL and Cypher.

Based on the transformation rules in Table 4, in the following we further give the transformation algorithms from SPARQL[t] to SPARQL (i.e., the algorithm 1 in **Table 5**) and from SPARQL to Cypher (i.e., the algorithm 2 in **Table 6**).

TABLE 5. The transformation algorithm from SPARQL[t] to SPARQL.

Algorithm 1. Transformation algorithm SPARQLT2SPARQL

```
INPUT: SPARQL[t] query
OUTPUT: SPARQL query
(1)  StringBuilder SPARQLString = new StringBuilder();
(2)  // save the transformed query language;
(3)  String whereFilterString = whereString.substring(new
     SparqlT2SparqlMatcher().SparqlT2SparqlMatcher(whereString,"
     FILTER"), whereString.length());
(4)  // use the following arraylist to store the transformed data in the
     RDFt graph;
(5)  ArrayList<ArrayList<String>> lsls = new
     ArrayList<ArrayList<String>>();
(6)  String [ ] statementRDFt = whereRDFtString.trim().split("\\.");
(7)  if(statementRDFt[i].contains("[")&&statementRDFt[i].contains("
     ])) then
(8)      lsls.add(transPredicate(statementRDFt[i].trim()));
         // transform triples with time information according to Table 4;
(9)  End if
(10) if(!(statementRDFt[i].contains("[")&&statementRDFt[i].contains(
     "]"))&&statementRDFt[i].contains("-"))
(11)     lsls.add(transPredicateOnlyn(statementRDFt[i].trim()));
         // transform triples without time information but with the
           update count n according to Table 4;
(12) End if
(13) if(!(statementRDFt[i].contains("[")&&statementRDFt[i].contains
     ("]"))||!(statementRDFt[i].contains("-")))
(14)     lsls.add(transPredicateRDF(statementRDFt[i]));
         // transform the ordinary triples according to Table 4;
(15) End if
(16) SPARQLString.append(prefixString).append("{"+whereStringS
     parql+"}").append(conditionSting);
(17) Return SPARQL query
```

As shown in Table 5, the algorithm first receives a string of an input SPARQL[t] query. Then the WHERE clause is separated and the different query forms are transformed according to the rules in Table 4. The transformed results are stored in a list, and a string of SPARQL query is finally returned.

TABLE 6. The transformation algorithm FROM SPARQL to Cypher.

Algorithm 2. Transformation algorithm SPARQL2Cypher

```
INPUT: SPARQL query
OUTPUT: Cypher query
(1)  Query query = QueryFactory.create(string);
(2)  Op op = Algebra.compile(query); // parse with JenaARQ;
(3)  String afterParse = new String(op.toString());
(4)  String[] brk = afterParse.split("triple");
(5)  For each i=1 to i=i+k in brk // process each string of brk;
(6)      if(i>1) then
(7)          matchStringp.append(",");
(8)      End if
(9)      matchStringp.append("(:"+ arr[0]+")-[");
(10)     For each j = i to j<i+5 in j
(11)         if(brk[i].contains(arr[1])) then // separate each string in
             brk[i] with a space into the new array;
(12)             if(pb[0].equals(arr[1])) then // select transformation
                 rules according to pb[1] and write to
                 matchStringpro;
(13)               matchStringpro.append(pb[1].substring(pb[1].lastI
                   ndexOf("#")+1,pb[1].length())+":"+pb[2].substring
                   (pb[2].lastIndexOf("#")+1,pb[2].length()));
(14)               k++;
(15)             else k=1;
(16)               matchStringpro.delete(0,matchStringpro.length());
(17)               matchStringpro.append(":"+arr[1]+"{");
(18)             End if
(19)         else k=1; break;
(20)         End if
(21)         matchStringp.append(matchStringpro).append("}]-
             >(:"+arr[2]+")");
(22)         getReturnString(returnS); // process the result set
(23)         String[] returnS = brk[0].split("filter");
(24)         returnString.append(getReturnString(returnS));
(25)         getWhereString(returnS[1]); //process where condition;
(26)         whereString.append(getWhereString(returnS[1]));
(27)         modifyString=modifyPart(string); //process modifier;
(28)         matchString.append(matchStringp);
(29)         cypherString.append(startString).append("MATCH
             "+matchString);
(30)         if(whereString.length()>0) then
(31)         cypherString.append("\nWHERE "+whereString);
(32)         End if
(33)         cypherString.append("\nRETURN "+returnString);
(34)         if(modifyString!=null) then
(35)         cypherString.append("\n"+modifyString.toString().trim
             ());
(36)         End if
(37)     End for
(38) End for
(39) Return Cypher query
```

After transforming SPARQL[t] into SPARQL, the following algorithm 2 in Table 6 further transforms the SPARQL to Cypher language. In Table 6, the algorithm first receives the string of the SPARQL query, uses Jena ARQ to parse the SPARQL, and handles each query part separately. Then, regarding to each triple, the algorithm gets the subject-predicate-object, makes the subject and object as nodes and predicate as relation, and adds the time and update count information into the relation. Also, each of FILTER conditions is transformed separately and connected by WHERE. The result modifiers in SPARQL are equivalently transformed into the modifiers in Cypher, e.g., the OFFSET in SPARQL is equivalent to the SKIP in Cypher. Final, the

variables in SPARQL are chosen as the return variables in Cypher, and multiple variables are separated by commas.

2) TRANSFORMATION OF BASIC GRAPH PATTERN

The basic graph pattern, which is used to represent constraints on RDFt query WHERE clauses, is the main component of the SAPRQL[t] query. In this section we further investigate and give the transformation algorithms from the SAPRQL[t] basic graph pattern to the SPARQL basic graph pattern and Cypher language. Regarding that whether each part in a triple of basic graph pattern is a constant value or a variable, the **Appendix 1** gives the transformation rules of each case in the basic graph pattern.

TABLE 7. The transformation algorithm of Basic Graph Pattern from SAPRQL[t] to SPARQL.

Algorithm 3 Transformation algorithm BaseRDFt2RDFPattern

INPUT: SAPRQL[t] basic graph pattern
OUTPUT: SAPRQL basic graph pattern

```
(1)  // Input a string of SAPRQL[t] basic graph pattern;
(2)  String rdftGraphPatternString = RDFtGraphPattern;
(3)  // Decompose rdftGraphPatternString to get each RDFt triple;
(4)  String [] statementRDFt = RDFtString.trim().split(" \\.");
(5)  // Process each triple separately; Retain the original ? or & and
     other variable symbols;
(6)  // Transform according to the rules in Appendix 1;
(7)  ArrayList<ArrayList<String>> lsls = new
     SparqlT2SparqlQuery().RDFt2RDFTriple(statementRDFt);
(8)  // Input the result of lsls into a string and output it;
(9)  For each i = 0 to i++ in lsls.size()
(10)   ArrayList<String> ls = new ArrayList<String>();
(11)   For each j = 0 to j++ in lsls.get(i).size()
(12)     sparqlPatternSTR.append(ls.get(i).get(j));
(13)   End for
(14) End for
```

TABLE 8. The transformation algorithm of Basic Graph Pattern from SAPRQL to Cypher.

Algorithm 4 Transformation algorithm BaseRDFPattern2Cypher

INPUT: SAPRQL basic graph patter
OUTPUT: Cypher graph

```
(1)  // Input a string of SAPRQL basic graph pattern;
(2)  // Store each triple in an array and process it separately;
(3)  String[] s = RDFGraphPattern.split(" \\.");
(4)  StringBuffer C1=new StringBuffer();
(5)  StringBuffer C2=new StringBuffer();
(6)  // Transform according to the rules in Appendix 1;
(7)  // Use the methods in the algorithm 2 to process the RDF triple
     pattern into a Cypher matching path form;
(8)  StringBuffer CypherStr = new
     Sparql2CypherQuery().MatchStringConstruct(C1,C2,s);
(9)  return CypherStr.toString()
```

Based on the transformation rules in the **Appendix 1**, the following **Table 7** and **Table 8** further give the corresponding transformation algorithms from SAPRQL[t] basic graph pattern to SPARQL basic graph pattern and Cypher graph supported in Neo4J graph database.

3) TRANSFORMATION of COMPLEX GRAPH PATTERNS

Based on the proposed basic graph pattern transformation rules and algorithms, this section decomposes the complex graph patterns of SAPRQL[t] and further transforms them into SPARQL complex graph patterns, and simultaneously transforms to Cypher graph patterns. The complex graph patterns include group graph pattern, optional graph pattern, and multi-graph union pattern. In the following we give the transformation rules and algorithms of each complex graph pattern.

- Group graph pattern

The group graph pattern in SPARQL[t] is the same as the group graph pattern in SPARQL, where two or more RDFt basic graph patterns are directly connected. For a SPARQL[t] query, all of the basic graph patterns in the group graph pattern must be matched. The transformation rules of the group graph pattern can be found in **Table 9.**

TABLE 9. Transformation of group graph pattern.

SPARQL[t] group graph pattern	SPARQL group graph pattern	Cypher group graph pattern
{S1, P1[ts1,te1]-n1,?O1} {S2, P2[ts2,te2]-n2,?O2}	{S1 ?p ?O1 . ?p rdf:type rdft:property . ?p rdft:hasStartTime ts1^^xsd:date . ?p rdft:hasEndTime te1^^xsd:date . ?p rdft:hasNumUpdate n1 . } { S1 ?p ?O1 . ?p rdf:type rdft:property . ?p rdft:hasStartTime ts2^^xsd:date . ?p rdft:hasEndTime te2^^xsd:date . ?p rdft:hasNumUpdate n2 . }	(S1)-[Relationship:p {type:'property', rdft_hasStartTime:'ts1 ^^xsd:date', rdft_hasEndTime:'te1^ ^xsd:date', rdft_hasNumUpdate:n1 }]→(O1) (S2)-[Relationship:p {type:'property', rdft_hasStartTime:'ts2 ^^xsd:date', rdft_hasEndTime:'te2^ ^xsd:date', rdft_hasNumUpdate:n2 }]→(O2)

- Optional graph pattern

The optional graph pattern in SPARQL[t] is to constrain an optional match to RDFt graph patterns by using the keyword OPTIONAL. Similarly, the keyword OPTIONAL is also supported in SPARQL and Cypher. The transformation rules of the optional graph pattern can be found in **Table 10**.

- Multiple graph pattern

The multiple graph pattern in SPARQL[t] uses the UNION keyword to connect multiple RDFt graph patterns, and the UNION keyword is also applied in SPARQL. In Cypher, the UNION or UNION ALL keyword performs a union operation on two result sets. The transformation rules of the multiple graph pattern can be found in **Table 11**.

Based on the transformation rules above, in the following we further give the corresponding transformation algorithms

TABLE 10. Transformation of optional graph pattern.

SPARQL[t] optional graph pattern	SPARQL optional graph pattern	Cypher optional graph pattern
{S1, P1[ts1,te1]-n1,?O1} OPTIONAL {S2, P2[ts2,te2]-n2,?O2}	{ S1 ?p ?O1 . ?p rdf:type rdft:property . ?p rdft:hasStartTime ts1^^xsd:date . ?p rdft:hasEndTime te1^^xsd:date . ?p rdft:hasNumUpdate n1 .} OPTIONAL { S2 ?p ?O2 . ?p rdf:type rdft:property . ?p rdft:hasStartTime ts2^^xsd:date . ?p rdft:hasEndTime te2^^xsd:date . ?p rdft:hasNumUpdate n2 .}	(S1)-[Relationship:p {type:'property', rdft_hasStartTime:'ts1^^xsd:date', rdft_hasEndTime:'te1^^xsd:date', rdft_hasNumUpdate:n1 }]→(O1) OPTIONAL MATCH (S2)-[Relationship:p {type:'property', rdft_hasStartTime:'ts2^^xsd:date', rdft_hasEndTime:'te2^^xsd:date', rdft_hasNumUpdate:n2 }]→(O2)

TABLE 11. Transformation of multiple graph pattern.

SPARQL[t] multiple graph pattern	SPARQL multiple graph pattern	Cypher multiple graph pattern
{S1, P1[ts1,te1]-n1,?O1} UNION {S2, P2[ts2,te2]-n2,?O2}	{ S1 ?p ?O1 . ?p rdf:type rdft:property . ?p rdft:hasStartTime ts1^^xsd:date . ?p rdft:hasEndTime te1^^xsd:date . ?p rdft:hasNumUpdate n1 .} UNION { S2 ?p ?O2 . ?p rdf:type rdft:property . ?p rdft:hasStartTime ts2^^xsd:date . ?p rdft:hasEndTime te2^^xsd:date . ?p rdft:hasNumUpdate n2 .}	(S1)-[Relationship:p {type:'property', rdft_hasStartTime:'ts1^^xsd:date', rdft_hasEndTime:'te1^^xsd:date', rdft_hasNumUpdate:n1}] →(O1) UNION/UNION ALL MATCH (S2)-[Relationship:p {type:'property', rdft_hasStartTime:'ts2^^xsd:date', rdft_hasEndTime:'te2^^xsd:date', rdft_hasNumUpdate:n2}] →(O2)

of complex graph patterns from SPARQL[t] to SPARQL and Cypher in **Table 12** and **Table 13**.

Based on the previous sections, the new temporal data representation model RDFt and its corresponding query language SPARQL[t] can represent and query the temporal data with both the time information and the update count information that widely exist in the real-world applications.

TABLE 12. Transformation algorithm of complex graph patterns from SPARQL[t] to SPARQL.

Algorithm 5 Translation algorithm ComplexRDFt2RDFPattern

INPUT: SAPRQL[t] complex graph patterns
OUTPUT: SPARQL complex graph patterns

```
(1)  // Input SAPRQL[t] complex graph patterns, and then decide
     the type of complex graph patterns;
(2)  Arraylist<String> list = new Arraylist<String> ();
(3)  if(SPARQLTString.contains("OPTIONAL"))
(4)    //Separate the parts with OPTIONAL keyword;
(5)    //Transform according to the rules in Table 10, and further
       transform each RDFt basic graph pattern in the OPTIONAL
       clause to RDF basic graph pattern according to the
       BaseRDFt2RDFPattern algorithm in Table 7;
(6)    BaseRDFt2RDFPattern(String RDFtGraphPatternStr);
(7)  else if(SPARQLTString.contains("UNION"))
(8)    //Separate the parts with UNION keyword;
(9)    //Transform according to the rules in Table 11, and further
       transform each RDFt basic graph pattern as similarly
       mentioned in the step (5) above;
(10) else if(isGroupGraph(SPARQLTString))
(11)   //Transform according to the rules in Table 9, and further
       transform each RDFt basic graph pattern as similarly
       mentioned in the step (5) above;
(12) End
```

TABLE 13. Transformation Algorithm of complex graph patterns from SPARQL to Cypher.

Algorithm 6 Translation algorithm ComplexRDFPattern2Cypher

INPUT: SPARQL complex graph patterns
OUTPUT: Cypher graph patterns

```
(1)  //Input SAPRQL[t] complex graph patterns, and then decide the type
     of complex graph patterns;
(2)  if(SPARQLString.contains("OPTIONAL"))
(3)    //Separate the parts with OPTIONAL keyword;
(4)    //Transform according to the rules in Table 10, and further
       transform each RDF basic graph pattern in the OPTIONAL
       clause to Cypher according to the BaseRDFPattern2Cypher
       algorithm in Table 8;
(5)    BaseRDFPattern2Cypher(String RDFGraphPattern);
(6)  else if(SPARQLString.contains("UNION"))
(7)    //Separate the parts with UNION keyword;
(8)    //Transform according to the rules in Table 11, and further
       transform each RDF basic graph pattern as similarly mentioned
       in the step (4) above;
(9)  else if(isGroupGraph(SPARQLTString))
(10)   //Transform according to the rules in Table 9, and further
       transform each RDF basic graph pattern as similarly mentioned
       in the step (4) above;
(11) End
```

V. PROTOTYPE AND EXPERIMENTS

Based on our proposed approaches, we further designed and implemented a prototype system, which includes two main functions: one is to represent and store the temporal data with both the time and update count information, and another one is to realize the query of the temporal data.

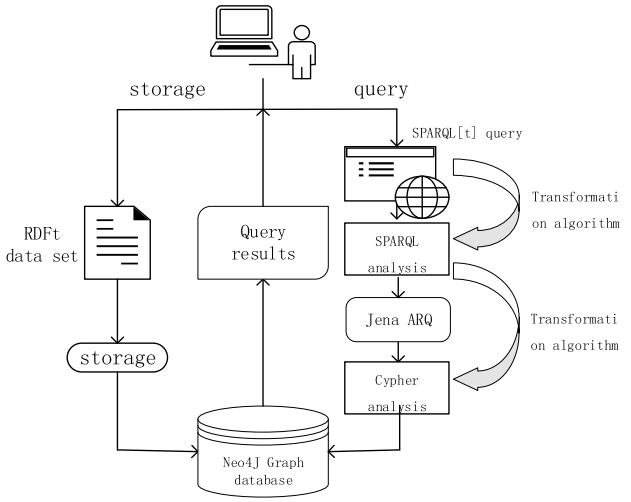


FIGURE 11. The overall architecture of the prototype.

In the following we *first* introduce the design and implementation of the prototype system (see Part *A*). *Then*, with two datasets we show that the proposed approach is feasible and the implemented system is efficient (see Part *B*).

A. THE ARCHITECTURE OF THE PROTOTYPE

The overall architecture of the prototype is shown in **Figure 11**. It mainly includes three modules:

- The storage module uses the Neo4J database [18] to store the RDFt data based on the storage algorithm as will be introduced in the following.
- The querying transformation module is to mainly transform the SPARQL[t] query language into the SPARQL and the Neo4J query language Cypher according to the query transformation algorithms proposed in the previous Part IV.
- The querying display module provides the query interface and displays the query results graphically.

The graphical user interface of the prototype system is shown in **Appendix 2**.

Moreover, as mentioned in the previous parts, the RDFt data proposed in this paper cannot be directly stored into Neo4J. Therefore, in the following we propose some rules and an algorithm for storing the RDFt data into the Neo4J database.

Rule 1: The subject of each RDFt triple is mapped to an Neo4J resource node with a label attribute as follows:

$$(S, P[ts, te]\text{-}n, O) \Rightarrow (:\text{Resource}\{\text{uri}:S\})$$

Rule 2: The predicate and object of each RDFt triple are mapped to a Neo4J relationship and a resource node with some label attributes as follows:

$$(S, P[ts, te]\text{-}n, O) \Rightarrow (:\text{Resource}\ \{\text{uri}:S\})\text{-}[\text{Relationship}: P\{\text{type:'property'}, \text{rdft\_hasStartTime:ts}, \text{rdft\_hasEndTime:te}, \text{rdft\_hasNumUpdate:}n\}]\text{-}(:\text{Resource}\ \{\text{uri}:O\})$$

TABLE 14. Algorithms for storing RDFt Data to Neo4J Database.

Algorithm 7 Storing algorithm RDFtToNeo4J

INPUT: RDFt data
OUTPUT: Data in Neo4J

```
(1)  //First, read the RDFt dataset into the model;
(2)  //The "subject-predicate-object" of each triple in the RDFt data
     model and the attributes of the predicate are separated;
(3)  while (iter.hasNext()) {
(4)    String subsubject = subject.substring(subject.lastIndexOf("/")
       +1, subject.length());
(5)    String subPredicate = predicate.substring(predicate.lastIndexOf
       ("/") +1, predicate.indexOf('['));
(6)    String hasStartTime = hasTime.substring(hasTime.indexOf('[')
       + 1, hasTime.indexOf(','));
(7)    String hasEndTime = hasTime.substring(hasTime.indexOf(',')
       + 1, hasTime.indexOf(']'));
(8)    String hasNumUpdate = predicate.substring(predicate.indexOf
       (']') + 2, predicate.length());
(9)    //Connect Neo4J database with JDBC
(10)   //Determine whether the node of subsubject exists in database;
(11)   if(isExistSubject(subsubject,statement)){
(12)     //Match to the node and add a new attribute relationship;
(13)   else if(isExistObject(object.toString(),statement)){
(14)     //Determine whether the node of object exists in database,
         and match to the node and add a new relationship as the node;
(15)   else //Create new nodes and relationships;
(16)   CypherString = getCypherStr(statement, subsubject, objectStr,
       predicateStr);
(17)   ResultSet resultSet = statement.executeQuery(CypherString);}
```

TABLE 15. Some main relations in NBA athlete dataset.

relations	number
hasBirthCity	3,519
Stature	4,579
Weight	4,573
High_School	3,855
Position	5,658
Plays_For	24,761
Rebound	24,761
Assist	24,761
Score	24,761

According to the rules above, **Table 14** further gives an algorithm for storing RDFt data into Neo4J database.

B. EXPERIMENTS

We use the following two datasets to carry out some storage and querying experiments (the figures in **Appendix 3** show the fragments of two datasets):

- One is the dataset of *NBA athletes*, which contains about 125,793 RDFt triples by crawling about 4,000 basic information of NBA stars from the NBA data website [21]. Table 15 shows the information of some main relations.
- Another is the dataset extracted from the YAGO2 [22], which is a spatially and temporally enhanced knowledge base from Wikipedia. In this paper, we mainly extract some temporal information and construct a RDFt

TABLE 16. A part of relations in our YAGO RDFt dataset.

relations	number
DiedIn	22,274
diedOnDate	315,528
happenedIn	5,192
happenedOnDate	22,039
occursSince/Until[2]	9,840
isLocatedIn	95,327[3]
livesIn	16,405
wasBornIn	56,415
wasBornOnDate	685,746
wasCreatedOnDate	467,194
wasDestoryedOnDate	24,218
playsFor	525,374
isAffiliatedTo	579,397

temporal dataset. Table 16 shows the information of parts of relations.

By means of our prototype, we store the two RDFt datasets into the Neo4J database and then carry out some queries. When a user input a SPARQL[t] query, which can be transformed into a corresponding SPARQL and the Neo4J query language Cypher, and then query results are returned. The detailed querying transformations and querying results on the two RDFt datasets are shown in the *Appendixes*:

- The **Appendix 4** shows some SPARQL[t] querying transformation examples and the corresponding querying results on the NBA dataset.
- The **Appendix 5** show some SPARQL[t] querying transformation examples and the corresponding results on the YAGO2 dataset.

The experimental results show that the proposed RDFt temporal data representation model, the query language SPARQL[t], and the query transformation algorithms can achieve the representation and querying requirements of the temporal data.

VI. RELATED WORK

Currently a huge amount of temporal RDF data is being proliferated and becoming available. As a result, efficient management of temporal RDF data is of increasing importance. The basic idea is to expand the time information based on RDF data model for different types of time information (such as time points and time intervals) and propose grammatical rules for extending the representation models and query languages.

In general, the existing work about the temporal extensions of RDF can be divided into three main extension forms:

- The *first* form is temporal data model based on the version control which is used to annotate the state of an RDF triple with the change of time [3].
- The *second* form is temporal data model based on the different extension forms of RDF triple (e.g., RDF quadtuple syntax). This kind of models define some new RDF syntaxes by extending the RDF triple to represent temporal information [23], [5], [19], [6], [14], [7], [13].
- The *third* form is temporal data model based on the original form of RDF triple by adding timestamp information. This kind of models re-define abstract semantics of the triple [9], [10], [8], [11], [24], [25], [12].

Regarding to the first form, in [3], a snapshot version-based RDF time series data model is proposed to track the changes in RDF repositories. The model is based on the assumption that an RDF statement is the smallest manageable piece of knowledge in an RDF repository. They argue that an RDF statement can only be added and removed. Each addition or removal turns the repository into a new state. The history of changes in the repository could be defined as sequence of states, as well, as a sequence of updates.

Regarding to the second form, in [23], a formal extension of RDF, called stRDF, for the representation and querying of linked geospatial data that changes over time. In stRDF, a temporal triple is a quad (s, p, o, t). In [5], [19], a temporal extension of RDF is proposed based on the idea of assigning timestamps to RDF triples. The model allows anonymous unknown timestamps in temporal RDF graphs. They also propose rules to translate temporal RDF into RDF graph. In [6], the authors extend the work of [5] to the case of indeterminate triples, and propose a temporal RDF model tRDF. The work in [14] presents an RDF based annotation framework for representing, reasoning and querying data on the Semantic Web. In particular, the framework supports statements annotated with fuzzy, temporal, and provenance, where an annotated triple is an expression $\tau : \lambda$, where τ is a triple and λ is an annotation value. The work in [7] presents a logic-based approach to represent and query validity time in RDF and OWL. Being similar to the idea in temporal RDF graph model [5], [19], the work in [13] proposes a formalism that is suitable to express temporal and non-temporal as well as probabilistic and non-probabilistic facts and constraints. Formally, a fact can be represented as: $conf\,(s, p, o)\,[t]$, where *conf* denotes the confidence value that the statement (s, p, o) is true, t is a time point or interval.

Regarding to the third form, in [9], an annotated RDF data model with annotation information is built on top of annotated logic, and the work mainly provides the semantics of extended partial ordered sets and supports the representation of uncertainty and time information in RDF triples. In [10], the timely YAGO is introduced, where they extend YAGO with temporal aspects by adding the concept of temporal facts. A temporal fact is a relation with an associated validity time. In [8], a temporal RDF graph pattern is proposed, and a method for representing the effective time that conforms to the existing RDF abstract model and semantics is given. In [11], the authors define a multi-temporal RDF triple $(s, p, o|T)$, where (s, p, o) is a standard triple subject, predicate, object and T is a timestamp assigning a temporal pertinence. In [24], an extension of the RDF model named TA-RDF model is presented in order to represent not only that

TABLE 17. ***Transformation rules of the basic graph pattern*** **from SAPRQL[t] to SPARQL and Cypher.**

Cases					Basic Graph Pattern		
s	***p***	***[ts, te]***	***-n***	***o***	***SPARQL[t]***	***SPARQL***	***Cypher***

×	√	√	√	√	SELECT ?s
					WHERE {
					?s p[ts,te]-n o .
					}
						SELECT ?s
						WHERE { ?s p o .
						p rdf:type rdf:property .
						p rdft:hasStartTime ts^^xsd:date .
						p rdft:hasEndTime te^^xsd:date .
						p rdft:hasNumUpdate n .}
							Match
							(s)-[:p{ type:property,
							hasStartTime:ts^^xsd:date,
							hasEndTime:te^^xsd:date,
							hasNumUpdate:n }]-(o)
							return s

×	√	×	√	√	SELECT ?s ?ts ?te
					WHERE {
					?s p[?ts,?te]-n o .
					}
						SELECT ?s ?ts ?te
						WHERE { ?s p o .
						p rdf:type rdf:property .
						p rdft:hasStartTime? ts .
						p rdft:hasEndTime ?te .
						p rdft:hasNumUpdate n .}
							Match
							(s)-[:p{ type:property,
							hasStartTime:ts,
							hasEndTime:te,
							hasNumUpdate:n }]-(o)
							return s,ts,te

×	√	√	×	√	SELECT ?s ?n
					WHERE {
					?s p[ts,te]-?n o .
					}
						SELECT ?s ?n
						WHERE { ?s p o .
						p rdf:type rdf:property .
						p rdft:hasStartTime ts^^xsd:date .
						p rdft:hasEndTime te^^xsd:date .
						p rdft:hasNumUpdate ?n . }
							Match
							(s)-[:p{ type:property,
							hasStartTime:ts,
							hasEndTime:te,
							hasNumUpdate:n }]-(o)
							return s,n

×	√	×	×	√	SELECT ?s ?ts ?te ?n
					WHERE {
					?s p[?ts,?te]-?n o .
					}
						SELECT ?s ?ts ?te ?n
						WHERE { ?s p o .
						p rdf:type rdf:property .
						p rdft:hasStartTime ?ts .
						p rdft:hasEndTime ?te .
						p rdft:hasNumUpdate ?n . }
							Match
							(s)-[:p{ type:property,
							hasStartTime:ts,
							hasEndTime:te,
							hasNumUpdate:n }]-(o)
							return s,ts,te,n

×	√	√	√	×	SELECT ?s ?o
					WHERE {
					?s p[ts,te]-n ?o .
					}
						SELECT ?s ?o
						WHERE { ?s p ?o .
						p rdf:type rdf:property .
						p rdft:hasStartTime ts^^xsd:date .
						p rdft:hasEndTime te^^xsd:date .
						p rdft:hasNumUpdate n . }
							Match
							(s)-[:p{ type:property,
							hasStartTime: ts^^xsd:date,
							hasEndTime: te^^xsd:date,
							hasNumUpdate:n }]-(o)
							return s,o

√	×	×	×	√	SELECT ?p ?ts ?te ?n
					WHERE {
					s ?p[?ts,?te]-?n o .
					}
						SELECT ?p ?ts ?te ?n
						WHERE { s ?p o .
						?p rdf:type rdf:property .
						?p rdft:hasStartTime ?ts .
						?p rdft:hasEndTime ?te .
						?p rdft:hasNumUpdate ?n . }
							Match
							(s)-[:p{ type:property,
							hasStartTime: ts,
							hasEndTime: te,
							hasNumUpdate:n }]-(o)
							return p

√	√	×	×	×	SELECT ?ts ?te ?n ?o
					WHERE {
					s p[?ts,?te]-?n ?o .
					}
						SELECT ?ts ?te ?n ?o
						WHERE { s p ?o .
						p rdf:type rdf:property .
						p rdft:hasStartTime ?ts .
						p rdft:hasEndTime ?te .
						p rdft:hasNumUpdate ?n . }
							Match
							(s)-[:p{ type:property,
							hasStartTime: ts,
							hasEndTime: te,
							hasNumUpdate:n }]-(o)
							return ts,te,n,o

√	√	×	√	×	SELECT ?ts ?te ?o
					WHERE {
					s p[?ts,?te]-n ?o .
					}
						SELECT ?ts ?te ?o
						WHERE { s p ?o .
						p rdf:type rdf:property .
						p rdft:hasStartTime ?ts .
						p rdft:hasEndTime ?te .
						p rdft:hasNumUpdate n . }
							Match
							(s)-[:p{ type:property,
							hasStartTime: ts,
							hasEndTime: te,
							hasNumUpdate:n }]-(o)
							return ts,te,o

√	√	√	×	×	SELECT ?n ?o
					WHERE {
					s p[ts,te]-?n ?o .
					}
						SELECT ?n ?o
						WHERE { s p ?o .
						p rdf:type rdf:property .
						p rdft:hasStartTime ts^^xsd:date .
						p rdft:hasEndTime te^^xsd:date .
						p rdft:hasNumUpdate ?n .}
							Match
							(s)-[:p{ type:property,
							hasStartTime: ts^^xsd:date,
							hasEndTime: te^^xsd:date,
							hasNumUpdate:n }]-(o)
							return n,o

Comments: × indicates that the item is a variable, and √ indicates that the item is a constant value or a resource.

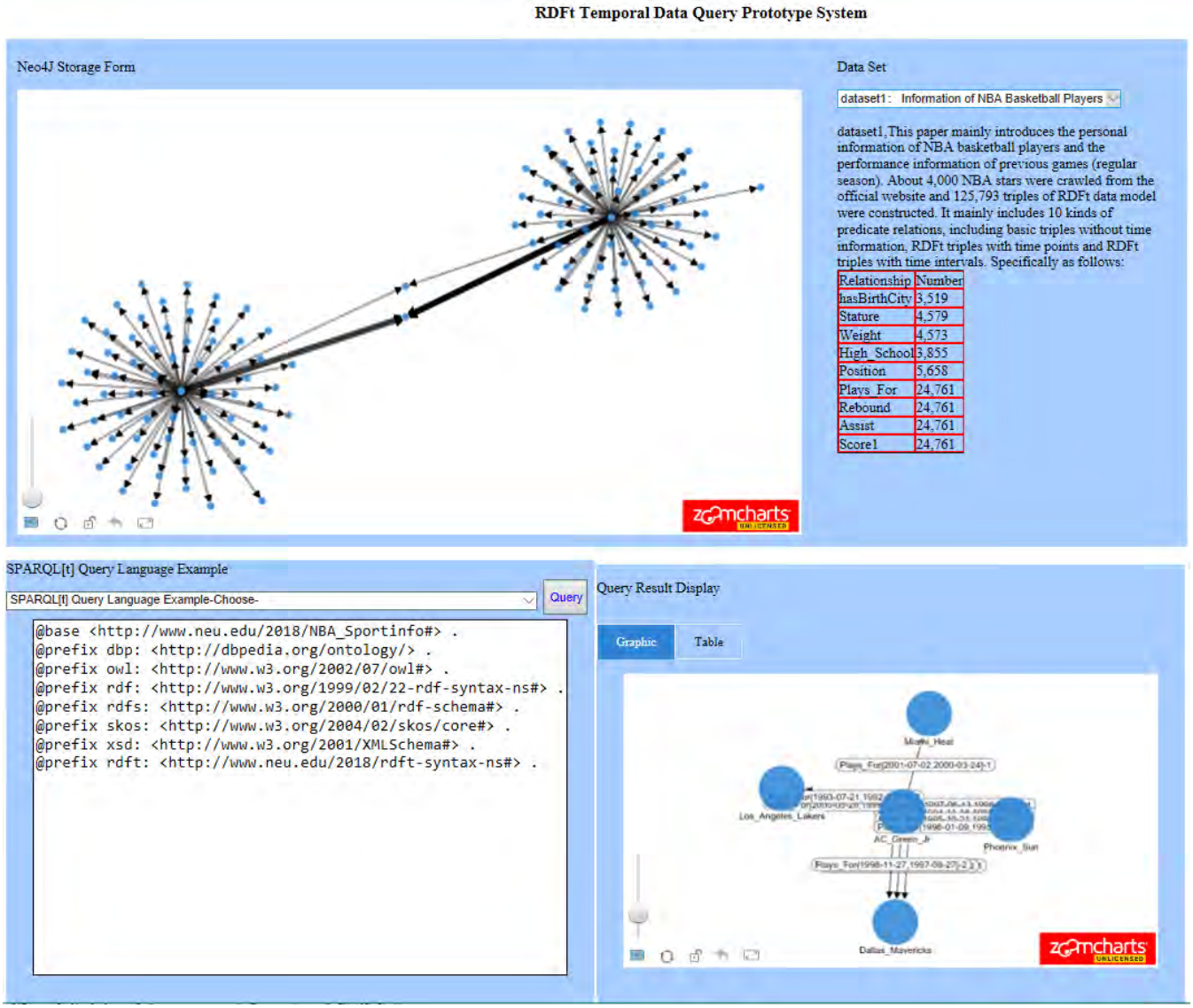


FIGURE 12. *Graphical User Interface* **of the prototype system.**

the properties between resources change over time, but that the resources themselves change. In [25], the authors propose a concept of temporal meta-information to represent time declaration and RDF graph data in large-scale linked data. In [12], an uncertain temporal knowledge base $KB =< F, C >$ is defined to resolve temporal conflicts in inconsistent RDF knowledge bases, where a fact in F has the form: $p\,(s, o, i)_{\mathrm{d}}$, where $p\,(s, o)$ is an RDF triple, i is a (half-open) temporal interval of the form $[t_{\mathrm{b}}, t_{\mathrm{e}})$, and $d \in [0, 1]$ is a confidence degree that $p\,(s, o)$ is true during interval i.

Furthermore, in order to query the time information from the temporal RDF data models as mentioned above, some corresponding temporal RDF query techniques are proposed. In brief, there are two main types, one is the SPARQL-like query languages by extending SPARQL with the temporal information. For example, the query languages AnQL [14], [7], stSPARQL [23], nSPARQL [26], SPARQL$^{\mathrm{T}}$ [27], [28], TA-SPARQL [24], [17], [11], EP-SPARQL [29], and CQELS (Continuous Query Evaluation over Linked Stream) [30]. Another is the specific query languages. For example, the query language of the temporal RDF graph model [5], which uses the expression method to describe the query statement; the five tuple query language in [6]; the atomic and connection query language in [9]; and the C-SPARQL query language for continuous data streams [31].

To our best knowledge, although there have been some researches on the representation and querying of temporal data as summarized above, there is no report about how to represent and query the temporal data with both the time information and the update count information. From the detailed summarization above, to sum up, it can be found obviously that there are several main differences between our

(1) *NBA athlete dataset*

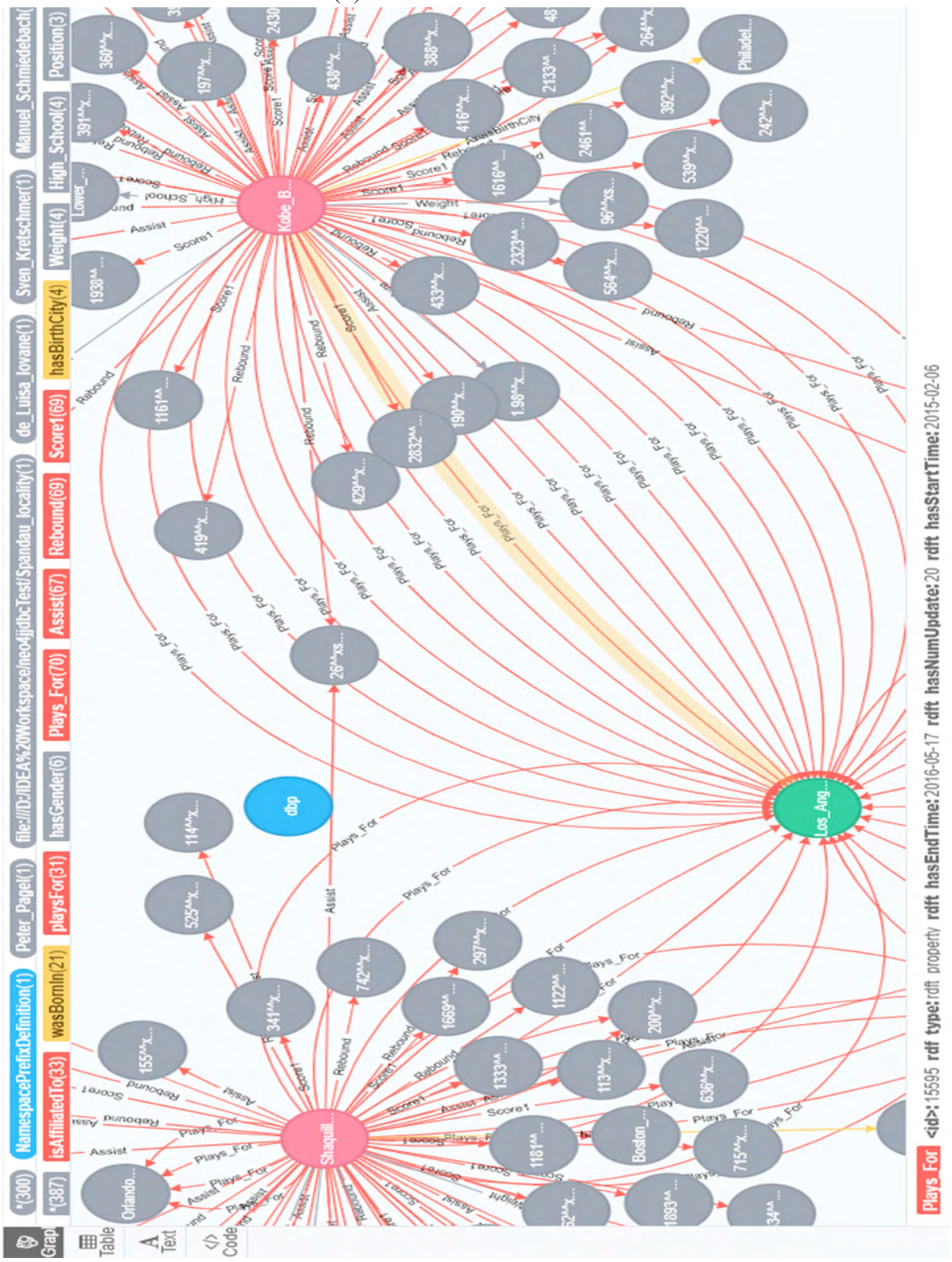


FIGURE 13. The fragments of *two datasets*: NBA athletes and YAGO.

(2) *YAGO dataset*

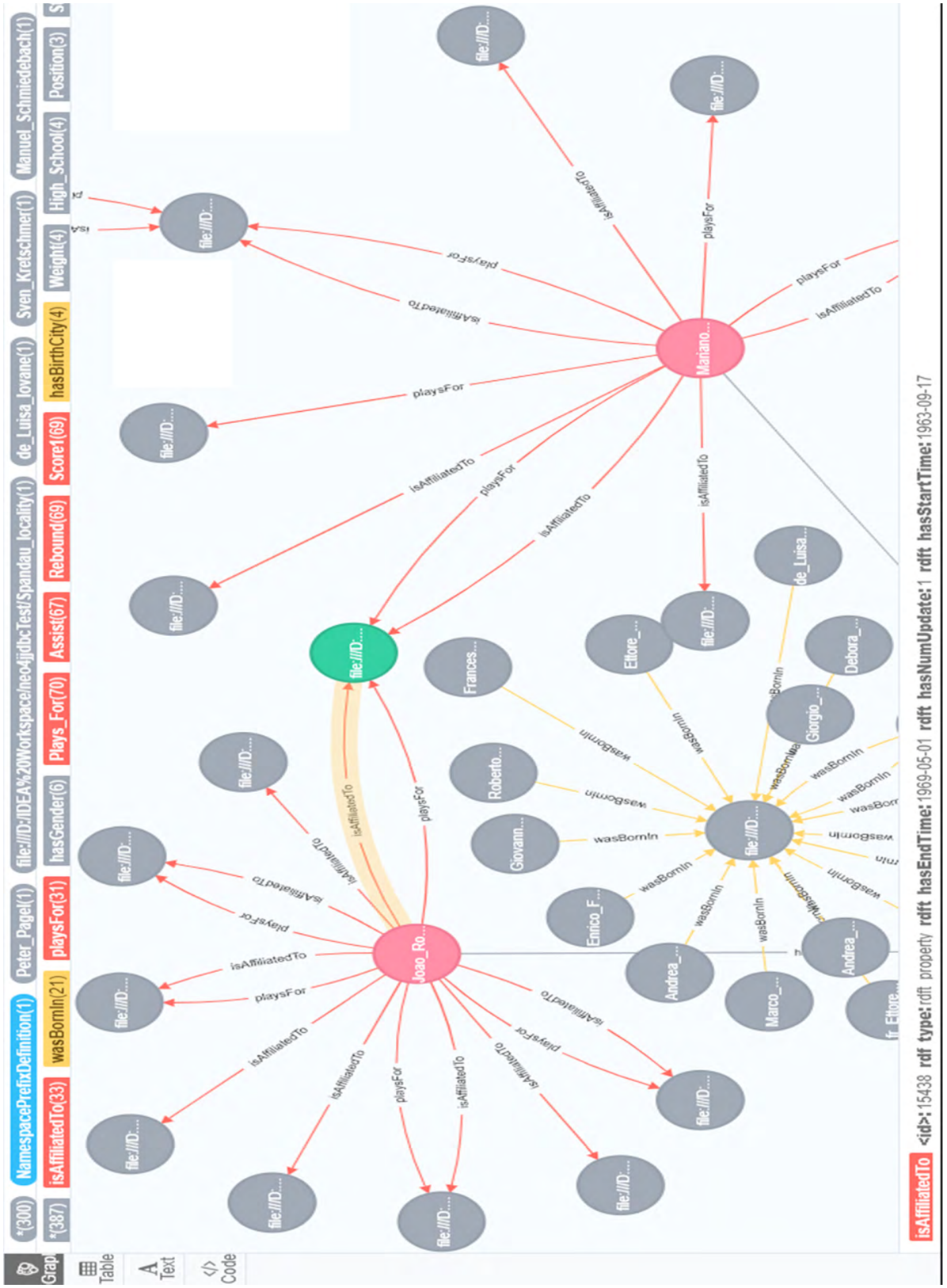


FIGURE 13. *(Continued.)* **The fragments of *two datasets*: NBA athletes and YAGO.**

TABLE 18. Some SPARQL[t] *querying transformation examples* and *querying results* on the NBA dataset.

(1) SPARQL[t] querying transformation examples on the NBA dataset

Queries	SPARQL[t]	SPARQL	Cypher
Q_1: Querying Yao Ming's Birth Date and Place.	SELECT ?t ?BirthCity WHERE { ?People info:hasBirthCity[?t]-1 ?BirthCity . FILTER ?People="Yao_Ming"}	SELECT ?t ?BirthCity WHERE { ?People ?hasBirthCity ?BirthCity . ?hasBirthCity a rdf:property . ?hasBirthCity rdft:hasTime ?t . ?hasBirthCity rdft:hasNumUpdate 1 . FILTER ?People="Yao_Ming" }	MATCH (People)-[Relationship0: hasBirthCity{type:'property', rdft_hasTime:t, rdft_hasNumUpdate:1}]→(Object) WHERE People="Yao_Ming" RETURN t,Object
Q_2: Querying A.C. Green's career information from 1992 to 2001.	SELECT ?ts ?te DISTINCT(?team) WHERE{?People info: Plays_For[?ts,?te]-?n ?team . FILTER ?ts ≥ "1992-01-01" and ?te ≤ "2001-01-01" FILTER ?People="AC_Green_Jr"}	SELECT ?ts ?te DISTINCT(?team) WHERE{ ?People ?Plays_For ?team . ?Plays_For a rdf:property . ?Plays_For rdft:hasStartTime ?ts . ?Plays_For rdft:hasEndTime ?te . ?Plays_For rdft:hasNumUpdate ?n . FILTER ?ts ≥ "1992-01-01" and ?te ≤ "2001-01-01" FILTER ?People="AC_Green_Jr"}	MATCH (AC_Green_Jr:AC_Green_Jr)- [Relationship:Plays_For]→(team) WHERE Relationship.rdft_hasStartTime ≥ '1992-01-01' and Relationship.rdft_hasEndTime ≤ '2001-01-01' RETURN AC_Green_Jr,Relationship,team

Q_3: Querying Shaquille-O'Neill's historical records and selecting the top 20 records.	SELECT ?s ?p ?ts ?te ?n ?o
	WHERE{
	?s info:Name ?name
	?s ?p[?ts,?te]-?n ?o .
	FILTER ?name
	="Shaquille_Rashaun_ONeal"
	}
	LIMIT 20
		SELECT ?s ?p ?ts ?te ?n ?o
		WHERE{ ?s info:Name ?name
		?s ?p ?o . ?p a rdf:property .
		?p rdft:hasStartTime ?ts .
		?p rdft:hasEndTime ?te .
		?p rdft:hasNumUpdate ?n .
		FILTER ?name ="Shaquille_
		Rashaun_ONeal"} LIMIT 20
			MATCH (Name)←
			[Relationship:Name]-(s)-
			[Relationship:p {type:'property',
			rdft_hasStartTime:ts,rdft_hasEndTime
			:te,rdft_hasNumUpdate:n}]→(o)
			WHERE Name ="Shaquille_Rashaun
			_ONeal" RETURN s,p,ts,te,n,o
			LIMIT 20

Q_4: Querying Allen Iverson's score in Philadelphia 76 and Denver Nuggets between 2006 and 2007.	SELECT ?Team ?Score
	WHERE{
	{"Allen_Ezail_Iverson"
	info:Plays_For[?ts1,?te1]-?n11
	?Team .
	"Allen_Ezail_Iverson"
	info:Score1[?ts1,?te1]-?n12
	?Score .
	FILTER ?Team
	="Philadelphia_76ers"
	FILTER ?ts1 ≥ "2006-01-01"
	and ?te1 ≤ "2007-01-01"}
	{"Allen_Ezail_Iverson"
	info:Plays_For[?ts2,?te2]-?n21
	?Team .
	"Allen_Ezail_Iverson"
	info:Score1[?ts2,?te2]-?n22
	?Score .
	FILTER ?Team
	="Denver_Nuggets"
	FILTER ?ts2 ≥ "2006-01-01"
	and ?te2 ≤ "2007-01-01"}
	}
		SELECT ?Team ?Score
		WHERE{{"Allen_Ezail_Iverson"
		?Plays_For ?Team. ?Plays_For a
		rdf:property .
		?Plays_For rdft:hasStartTime ?ts1 .
		?Plays_For rdft:hasEndTime ?te1 .
		?Plays_For rdft:hasNumUpdate ?n11 .
		"Allen_Ezail_Iverson" ?Score1 ?Score .
		?Score1 a rdf:property .
		?Score1 rdft:hasStartTime ?ts1 .
		?Score1 rdft:hasEndTime ?te1 .
		?Score1 rdft:hasNumUpdate ?n12 .
		FILTER ?Team ="Philadelphia_76ers"
		FILTER ?ts1 ≥"2006-01-01" and
		?te1≤"2007-01-01"}
		{"Allen_Ezail_Iverson" ?Plays_For
		?Team . ?Plays_For a rdf:property .
		?Plays_For rdft:hasStartTime ?ts2 .
		?Plays_For rdft:hasEndTime ?te2 .
		?Plays_For rdft:hasNumUpdate ?n21 .
		"Allen_Ezail_Iverson"
		info:Score1[?ts2,?te2]-?n22 ?Score .
		?Plays_For a rdf:property .
		?Plays_For rdft:hasStartTime ?ts2 .
		?Plays_For rdft:hasEndTime ?te2 .
		?Plays_For rdft:hasNumUpdate ?n22 .
		FILTER ?Team ="Denver_Nuggets"
		FILTER ?ts2≥"2006-01-01" and
		?te2≤"2007-01-01"}}
			MATCH (Score)←
			[Relationship0:Score1 {type:'property',
			rdft_hasStartTime:ts1,rdft_hasEndTi
			me:te1,rdft_hasNumUpdate:n12}]-
			(People: "Allen_Ezail_Iverson")-
			[Relationship0: Plays_For
			{type:'property',rdft_hasStartTime:ts1,
			rdft_hasEndTime:te1,rdft_hasNumUp
			date:n11}]→(Team:
			"AC_Green_Jr"),
			(Score)←
			[Relationship1:Score1 {type:'property',
			rdft_hasStartTime:ts2,rdft_hasEndTi
			me:te2,rdft_hasNumUpdate:n22}]-
			(People: "Allen_Ezail_Iverson")-
			[Relationship1: Plays_For
			{type:'property',rdft_hasStartTime:ts2,
			rdft_hasEndTime:te2,rdft_hasNumUp
			date:n21}]→(Team: "AC_Green_Jr"),
			WHERE (Relationship0.ts1≥"2006-
			01-01" and Relationship0.te1≤"2007-
			01-01")
			AND ((Relationship1.ts2≥"2006-01-
			01" and Relationship1.te2≤"2007-01-
			01")
			)
			RETURN Team, Score

Q_5: Querying the height information of athletes, and their birth	SELECT ?People ?num ?City ?t	SELECT ?People ?num ?City ?t WHERE{ {?People info:Stature ?num .}	MATCH (People)- [Relationship0:Stature]→(num)

TABLE 18. ***(Continued.)*** **Some SPARQL[t]** ***querying transformation examples*** **and** ***querying results*** **on the NBA dataset.**

places are chosen as the optional attribute (OPTIONAL). The result set is constrained to 5-15.	WHERE{ {?People info:Stature ?num .} OPTIONAL {?People info:hasBirthCity[?t]-?n ?City .}} OFFSET 5 LIMIT 15	OPTIONAL {?People ?hasBirthCity ?City . ?hasBirthCity a rdf:property . ?hasBirthCity rdft:hasTime ?t . ?hasBirthCity rdft:hasNumUpdate ?n.}} OFFSET 5 LIMIT 15	OPTIONAL MATCH (People)-[Relationship1: hasBirthCity {type:'property', rdft_hasTime:t, rdft_hasNumUpdate:n}]→(City) RETURN People,num,City,t SKIP 5 LIMIT 15
Q_6: Querying Lin Shuhao's highest total score on the Rockets or the Lakers (*UNION*).	SELECT (MAX(?Grade1) AS ?Score) (MAX(?Grade2) AS ?Score) WHERE{ {"Jeremy_ShuHow_Lin" info:Plays_For[?ts1,?te1]-?n11 > "Houston_Rockets" . "Jeremy_ShuHow_Lin" info:Score1[?ts1,?te1]-?n12 > ?Grade1 .} UNION {"Jeremy_ShuHow_Lin" info:Plays_For[?ts2,?te2]-?n21> "Los_Angeles_Lakers" . "Jeremy_ShuHow_Lin" info:Score1[?ts2, ?te2]-?n22 > ?Grade2 .} }	SELECT (MAX(?Grade1) AS ?Score) (MAX(?Grade2) AS ?Score) WHERE{ {"Jeremy_ShuHow_Lin" ?Plays_For "Houston_Rockets" . ?Plays_For a rdf:property . ?Plays_For rdft:hasStartTime ?ts1 . ?Plays_For rdft:hasEndTime ?te1 . ?Plays_For rdft:hasNumUpdate ?n11 . "Jeremy_ShuHow_Lin" ?Score1 ?Grade1. ?Score1 a rdf:property . ?Score1 rdft:hasStartTime ?ts1 . ?Score1 rdft:hasEndTime ?te1 . ?Score1 rdft:hasNumUpdate ?n12 .} UNION {"Jeremy_ShuHow_ Lin" ?Plays_For "Los_Angeles_Lakers". ?Plays_For a rdf:property . ?Plays_For rdft:hasStartTime ?ts2 . ?Plays_For rdft:hasEndTime ?te2 . ?Plays_For rdft:hasNumUpdate ?n21 . "Jeremy_ShuHow_Lin" ?Score1 ?Grade2. ?Score1 a rdf:property . ?Score1 rdft:hasStartTime ?ts2 . ?Score1 rdft:hasEndTime ?te2 . ?Score1 rdft:hasNumUpdate ?n22 .} }	MATCH (Grade1)←[Relationship:?Score1{type:'property', rdft_hasStartTime:ts1,rdft_hasEndTime:te1,rdft_hasNumUpdate:n12}]-(People: "Jeremy_ShuHow_Lin")-[Relationship: Plays_For {type:'property',rdft_hasStartTime:ts1, rdft_hasEndTime:te1,rdft_hasNumUpdate:n11}]→(Team: "Houston_Rockets") RETURN max(Grade1) UNION MATCH (Grade2)←[Relationship:?Score1{type:'property', rdft_hasStartTime:ts2,rdft_hasEndTime:te2,rdft_hasNumUpdate:n22}]-(People: "Jeremy_ShuHow_Lin"))-[Relationship: Plays_For {type:'property',rdft_hasStartTime:ts2, rdft_hasEndTime:te2,rdft_hasNumUpdate:n21}]→ (Team: "Los_Angeles_Lakers") RETURN max(Grade2)

Q_7: Querying whether Bryant played for the Lakers from 1996 to 2016.	ASK {
	"Kobe_Bean_Bryant"
	info:Plays_For[?ts,?te]-?n "Los_Angeles_Lakers" .
	FILTER ?ts ≥ "1996-01-01" and ?te ≤ "2016-12-30"
	}
		ASK {"Kobe_Bean_Bryant"
		?Plays_For "Los_Angeles_Lakers".
		?Plays_For rdf:type rdft:property .
		?Plays_For rdft:hasStartTime ?ts^^xsd:date . ?Plays_For rdft:hasEndTime ?te^^xsd:date . ?Plays_For rdft:hasNumUpdate 1 . FILTER ?ts ≥ "1996-01-01" and ?te ≤ "2016-12-30"}
			MATCH (Kobe_Bean_Bryant)-[Relationship:Plays_For]→(Los_Angeles_Lakers) WHERE Relationship.rdft_hasStartTime≥'1996-01-01' and Relationship.rdft_hasEndTime≤'2016-12-30' RETURN [Relationship]

Q_8: Constructing a new relation *info:isTeammate* to represent that two athletes are teammates.	CONSTRUCT {"Kobe_Bean_Bryant" info:isTeammate[?ts1,?te1]-1 "Shaquille_Rashaun_ONeal" .} WHERE{ "Kobe_Bean_Bryant" info:Plays_For[?ts1,?te1]-?n1 "Los_Angeles_Lakers" . "Shaquille_Rashaun_ONeal" info:Plays_For[?ts2,?te2]-?n2 "Los_Angeles_Lakers" . FILTER (?ts1 ≥ ?ts2 and ?te1 ≤ ?te2) or (?ts1 ≤ ?ts2 and ?te1 ≥ ?te2)}	CONSTRUCT {"Kobe_Bean_Bryant" ?isTeammate "Shaquille_Rashaun_ONeal" . ?isTeammate a rdf:property . ?isTeammate rdft:hasStartTime ?ts1 . ?isTeammate rdft:hasEndTime ?te1 . ?isTeammate rdft:hasNumUpdate 1 .} WHERE{ "Kobe_Bean_Bryant" ?Plays_For "Los_Angeles_Lakers" . ?Plays_For a rdf:property . ?Plays_For rdft:hasStartTime ?ts1 . ?Plays_For rdft:hasEndTime ?te1 . ?Plays_For rdft:hasNumUpdate ?n1 . "Shaquille_Rashaun_ONeal" ?Plays_For "Los_Angeles_Lakers" . ?Plays_For a rdf:property . ?Plays_For rdft:hasStartTime ?ts2 . ?Plays_For rdft:hasEndTime ?te2 . ?Plays_For rdft:hasNumUpdate ?n2 . FILTER (?ts1 ≥ ?ts2 and ?te1 ≤ ?te2) or (?ts1 ≤ ?ts2 and ?te1 ≥ ?te2)}	MATCH (Kobe_Bean_Bryant:Kobe_Bean_Bryant)-[Relationship: Plays_For {type:'property',rdft_hasStartTime:ts1, rdft_hasEndTime:te1,rdft_hasNumUpdate:n1}]→(objects:Los_Angeles_Lakers)←[Relationship: Plays_For {type:'property',rdft_hasStartTime:ts2, rdft_hasEndTime:te2,rdft_hasNumUpdate:n2}]-(Shaquille_Rashaun_ONeal: Shaquille_Rashaun_ONeal) WHERE (Relationship.ts1 ≥ Relationship0.ts2 and Relationship0.te1 ≤ Relationship.te2) or (Relationship0.ts1 ≤ Relationship0.ts2 and Relationship.te1 ≥ Relationship.te2) RETURN ts1,te1; CREATE (Kobe_Bean_Bryant:Kobe_Bean_Bryant)-[Relationship: isTeammate {type:'property',rdft_hasStartTime:ts1, rdft_hasEndTime:te1,rdft_hasNumUpdate:1}]-(Shaquille_Rashaun_ONeal: Shaquille_Rashaun_ONeal)

TABLE 18. ***(Continued.)*** **Some SPARQL[t] *querying transformation examples* and *querying results* on the NBA dataset.**

Q_9: Querying all players' personal information of the 2018 Warriors.	DESCRIBE ?People {?People info:Plays_For[?ts,?te]-?n "Golden_State_Warriors" . FILTER ?ts ≥ "2017-12-30" and ?te ≤ "2018-12-30" }	DESCRIBE ?People {?People ?Plays_For "Golden_State_Warriors" . ?Plays_For a rdf:property . ?Plays_For rdft:hasStartTime ?ts . ?Plays_For rdft:hasEndTime ?te . ?Plays_For rdft:hasNumUpdate ?n . FILTER ?ts ≥ "2017-12-30" and ?te ≤ "2018-12-30"}	MATCH (People)-[Relationship0: Plays_For {type:'property', rdft_hasStartTime:ts,rdft_hasEndTime :te,rdft_hasNumUpdate:n}]→(Team: "Golden_State_Warriors") WHERE Relationship0.ts ≥ "2017-12-30" and Relationship0.te ≤ "2018-12-30" RETURN (People)-[0…5]→(m)

(2) SPARQL[t] querying results of Q_1-Q_9 on the NBA dataset

Queries	Querying results
Q_1	"1980-9-12" "Shanghai_China"
Q_2	[1992-03-15,1993-07-21]-8 "Los_Angeles_Lakers" . [1993-01-23,1994-11-16]-1 "Phoenix_Sun" . [1994-08-17,1995-10-21]-2 "Phoenix_Sun" . [1995-03-21,1996-01-09]-3 "Phoenix_Sun" . [1996-01-25,1997-06-13]-4 "Phoenix_Sun" . [1996-03-07,1997-03-10]-1 "Dallas_Mavericks" . [1997-08-27,1998-11-27]-2 "Dallas_Mavericks" . [1998-03-22,1999-01-10]-3 "Dallas_Mavericks" . [1999-02-08,2000-05-26]-9 "Los_Angeles_Lakers" . [2000-03-24,2001-07-02]-1 "Miami_Heat" .
Q_3	Return to the top 20 records of Shaquille O'Neill's historical competition information. Some of the data are as follows: "Shaquille_Rashaun_ONeal" hasBirthCity[1972-03-06]-1 "Newark_New_Jersey" . "Shaquille_Rashaun_ONeal" Stature "2.16"^^<xsd:decimal> . "Shaquille_Rashaun_ONeal" Weight "147"^^<xsd:integer> .
Q_4	"Philadelphia_76ers" 468 "Denver_Nuggets" 1241
Q_5	Includes the height of the athlete and the city and time of birth (optional attributes). Some of the data are as follows: "Anthony_Bennett" "Toronto_Ontario" "1993-3-14" "Anthony_Bennett" "2.03"^^<xsd:decimal> "Anthony_Brown" "1.96"^^<xsd:decimal>
Q_6	1095
Q_7	TRUE/FALSE
Q_8	The results are stored in Neo4J database
Q_9	All triple sets with ?People as the subject

work in this paper and the existing works: (*i*) Regarding to the representation of temporal data, as has mentioned in the introduction of Part I, each of the existing representation models is proposed for different representation demands of temporal data, and the existing work are not enough to represent and manage all types of temporal information in practical applications (e.g., the temporal data with both the time information and the update count information). Therefore, in this paper we propose a new temporal RDF data model called RDFt, which can represent the temporal data with both the time information and the update count information, and we also present the syntax and semantics of RDFt model in detail and provide the example to well explain the model; (*ii*) Regarding to the query of temporal data, the existing query languages cannot realize the query for our proposed RDFt data model. To this end, we propose a query language called SPARQL[t] for RDFt, and we also propose the querying transformation algorithms from SPARQL[t] to SPARQL and Cypher.

TABLE 19. Some SPARQL[t] *querying transformation examples* and *querying results* on the YAGO dataset.

(1) SPARQL[t] querying transformation examples on the YAGO dataset

Queries	SPARQL[t]	SPARQL	Cypher
Q_1: Querying Fabio_Ongaro's birth date and place.	SELECT ?t ?BirthCity WHERE { ?People wasBornIn [?t]-1 ?BirthCity . FILTER ?People = "Fabio_Ongaro"}	SELECT ?t ?BirthCity WHERE {?People ?wasBornIn ?BirthCity . ?wasBornIn a rdf:property . ?wasBornIn rdft:hasTime ?t . ?wasBornIn rdft:hasNumUpdate 1 . FILTER ?People="Fabio_Ongaro" }	MATCH (People)-[Relationship: wasBornIn {type:'property', rdft_hasTime:T,rdft_hasNumUpdate:1}]→ (Object) WHERE People="Fabio_Ongaro" RETURN T, Object
Q_2: Querying Mauricio_Soler's career information from 2001 to 2006.	SELECT ?ts ?te DISTINCT(?team) WHERE { ?People info:playsFor [?ts,?te]-?n ?team . FILTER ?ts ≥ "2001-01-01" and ?te≤"2006-01-01" FILTER ?People="Mauricio_ Soler"}	SELECT ?ts ?te DISTINCT(?team) WHERE{ ?People ?playsFor ?team . ?playsFor a rdf:property . ?playsFor rdft:hasStartTime ?ts . ?playsFor rdft:hasEndTime ?te . ?playsFor rdft:hasNumUpdate ?n . FILTER ?ts≥"2001-01-01" and ?te≤"2006-01-01" FILTER ?People="Mauricio_Soler"}	MATCH (People)-[Relationship: playsFor {type:'property',rdft_hasStartTime:Ts,rdft_ hasEndTime:Te,rdft_hasNumUpdate:N}] →(Team) WHERE Relationship.Ts≥'2001-01-01' and Relationship.Te≤'2006-01-01' WHERE People="Mauricio_Soler" RETURN Ts, Te, Team
Q_3: Querying the information of Ricardo Souza Silva and selecting the first 15 records.	SELECT ?s ?p ?ts ?te ?n ?o WHERE{ ?s ?p[?ts,?te]-?n ?o . FILTER ?s ="Ricardo_Souza_Silva" } LIMIT 15	SELECT ?s ?p ?ts ?te ?n ?o WHERE{ ?s ?p ?o . ?p a rdf:property . ?p rdft:hasStartTime ?ts . ?p rdft: hasEndTime ?te . ?p rdft:hasNumUpdate ?n . FILTER ?s ="Ricardo_Souza_Silva"} LIMIT 15	MATCH (Name)←[Relationship:Name]- (S)-[Relationship:p {type:'property', rdft_hasStartTime:Ts,rdft_hasEndTime:Te, rdft_hasNumUpdate:N}]→(O) WHERE Name ='Ricardo_Souza_Silva' RETURN S, P, Ts, Te, N, O LIMIT 15
Q_4: Querying the team of athletes who lived in Falkirk in 1988.	SELECT DISTINCT(?Team) WHERE{ {?People wasBornIn[?t]-1 "Falkirk" . FILTER ?t ≥ "1988-01-01" and ?t ≤ "1988-12-31" } {?People playsFor[?ts1,?te1]-?n1 ?Team .}}	SELECT DISTINCT(?Team) WHERE{ {?People ?wasBornIn "Falkirk" . ?wasBornIn a rdf:property . ?wasBornIn rdft:hasTime ?t . ?wasBornIn rdft:hasNumUpdate 1 . FILTER ?t ≥ "1988-01-01" and ?t ≤ "1988-12-31" } {?People ?playsFor ?Team . ?playsFor a rdf:property . ?playsFor rdft:hasStartTime ?ts1 . ?playsFor rdft:hasEndTime ?te1 . ?playsFor rdft:hasNumUpdate ?n1 .}}	MATCH (People)-[Relationship0: wasBornIn{type:'property',rdft_hasTime:T, rdft_hasEndTime:Te1,rdft_hasNumUpdate: 1}]→(Team: 'Falkirk'), (People)-[Relationship1:playsFor {type:'property',rdft_hasStartTime:Ts1,rdft _hasEndTime:Te1,rdft_hasNumUpdate:N1 }]→(Team) WHERE Relationship0. rdft_hasTime ≥ '1988-03-31' and Relationship0. rdft_hasTime ≤ '1988-12-31' RETURN Team
Q_5: Querying the information of male athletes, and their birth places are chosen as the optional attribute (OPTIONAL). The result set is constrained to 5-15.	SELECT ?People ?City WHERE{ {?People hasGender <male> .} OPTIONAL {?People wasBornIn [?t]-?n ?City .}} OFFSET 5 LIMIT 15	SELECT ?People ?City WHERE{ {?People hasGender <male> .} OPTIONAL {?People ?wasBornIn ?City . ?wasBornIn a rdf:property . ?wasBornIn rdft:hasTime ?t . ?wasBornIn rdft:hasNumUpdate ?n .}} OFFSET 5 LIMIT 15	MATCH (People)-[Relationship: hasGender]→(Male) OPTIONAL MATCH (People)- [Relationship: wasBornIn {type:'property',rdft_hasTime:T,rdft_hasNu mUpdate:N}]→(City) SKIP 5 LIMIT 15
Q_6: Querying the teams that Gustavo_ Bou plays for and belongs to.	SELECT DISTINCT(?Team) WHERE{ {"Gustavo_Bou" playsFor[?ts1,?te1]-?n ?Team .} UNION {"Gustavo_Bou" isAffiliatedTo [?ts2,?te2]-?n ?Team .} } FILTER ?ts1=?ts2 and ?te1=?te2	SELECT DISTINCT(?Team) WHERE{{"Gustavo_Bou" ?playsFor ?Team. ?Plays_For a rdf:property . ?Plays_For rdft:hasStartTime ?ts1 . ?Plays_For rdft:hasEndTime ?te1 . ?Plays_For rdft:hasNumUpdate ?n .} UNION {"Gustavo_Bou" ?isAffiliatedTo ?Team . ?isAffiliatedTo a rdf:property . ?isAffiliatedTo rdft:hasStartTime ?ts2 . ?isAffiliatedTo rdft:hasEndTime ?te2 . ?isAffiliatedTo rdft:hasNumUpdate ?n .}} FILTER ?ts1=?ts2 and ?te1=?te2	(People: 'Gustavo_Bou')-[Relationship0: Plays_For {type:'property', rdft_hasStartTime:Ts1,rdft_hasEndTime:T e1,rdft_hasNumUpdate:N}]→(Team) UNION MATCH (People: 'Gustavo_Bou') -[Relationship1: isAffiliatedTo {type: 'property',rdft_hasStartTime:Ts2,rdft_hasE ndTime:Te2,rdft_hasNumUpdate:N}]→ (Team) WHERE Relationship0. rdft_hasStartTime = Relationship1. rdft_hasStartTime and Relationship0. rdft_hasEndTime = Relationship1. rdft_ hasEndTime RETURN DISTINCT(Team)
Q_7: Querying whether the The_Culture plays for the team Israel_ national_football_team from 1996 to 2005.	ASK {"The_Culture" info:playsFor[?ts,?te]-?n "Israel_national_football_team" . FILTER ?ts ≥ "1996-01-01" and ?te ≤ "2005-12-30" }	ASK { "The_Culture" ?playsFor "Israel_national_football_team" . ?playsFor rdf:type rdft:property . ?playsFor rdft:hasStartTime ?ts^^xsd:date . ?playsFor rdft:hasEndTime ?te^^xsd:date . ?playsFor rdft:hasNumUpdate 1 . FILTER ?ts ≥ '1996-01-01' and ?te ≤ '2005-12-30'}	MATCH (The_Culture)- [Relationship:playsFor]→(Israel_national_f ootball_team) WHERE Relationship.rdft_hasStartTime ≥ '1996-01-01' and Relationship.rdft_hasEndTime ≤ '2005-12-30' RETURN [Relationship]

TABLE 19. ***(Continued.)*** **Some SPARQL[t]** ***querying transformation examples*** **and** ***querying results*** **on the YAGO dataset.**

Q_8: Querying all players' personal information of Spandauer_SV in 2018.	DESCRIBE ?People {?People info:playsFor[?ts,?te]-?n "Spandauer_SV" . FILTER ?ts ≥ '2017-12-30' and ?te ≤ '2018-12-30' }	DESCRIBE ?People {?People ?playsFor "Spandauer_SV" . ?playsFor a rdf:property . ?playsFor rdft:hasStartTime ?ts . ?playsFor rdft:hasEndTime ?te . ?playsFor rdft:hasNumUpdate ?n . FILTER ?ts ≥ '2017-12-30' and ?te ≤ '2018-12-30'}	MATCH (People)-[Relationship: playsFor {type:'property',rdft_hasStartTime:Ts,rdft_hasEndTime:Te,rdft_hasNumUpdate:N}] →(Team: 'Spandauer_SV') WHERE Relationship.Ts ≥ '2017-12-30' and Relationship.Te ≤ '2018-12-30' RETURN (People)-[0…5]→(m)
Q_9: Constructing a new teammate relationship called *isTeammate* between two athletes	CONSTRUCT {"Mariano_Pavone" isTeammate[?ts1,?te1]-1 "Joao_Rojas" .} WHERE{ "Mariano_Pavone" isAffiliatedTo [?ts1,?te1]-?n1 "Cruz_Azul" . "Joao_Rojas" isAffiliatedTo [?ts2,?te2]-?n2 "Cruz_Azul" . FILTER (?ts1 ≥ ?ts2 and ?te1 ≤ ?te2) or (?ts1 ≤ ?ts2 and ?te1 ≥ ?te2)}	CONSTRUCT {"Mariano_Pavone" ?isTeammate "Joao_Rojas". ?isTeammate a rdf:property . ?isTeammate rdft:hasStartTime ?ts1 . ?isTeammate rdft:hasEndTime ?te1 . ?isTeammate rdft:hasNumUpdate 1 .} WHERE{ "Mariano_Pavone" ?isAffiliatedTo "Cruz_Azul" . ?isAffiliatedTo a rdf:property . ?isAffiliatedTo rdft:hasStartTime ?ts1 . ?isAffiliatedTo rdft:hasEndTime ?te1 . ?isAffiliatedTo rdft:hasNumUpdate ?n1 . "Joao_Rojas" ?isAffiliatedTo "Cruz_Azul" . ?isAffiliatedTo a rdf:property . ?isAffiliatedTo rdft:hasStartTime ?ts2 . ?isAffiliatedTo rdft:hasEndTime ?te2 . ?isAffiliatedTo rdft:hasNumUpdate ?n2 . FILTER (?ts1 ≥ ?ts2 and ?te1 ≤ ?te2) or (?ts1 ≤ ?ts2 and ?te1 ≥ ?te2)}	MATCH (Mariano_Pavone: Mariano_Pavone)-[Relationship: isAffiliatedTo {type:'property', rdft_hasStartTime:Ts1,rdft_hasEndTime:Te1,rdft_hasNumUpdate:N1}]→(objects: Cruz_Azul)←[Relationship: isAffiliatedTo {type:'property',rdft_hasStartTime:Ts2,rdft_hasEndTime:Te2,rdft_hasNumUpdate:N2}]-(Joao_Rojas: Joao_Rojas) WHERE (Relationship.ts1 ≥ Relationship0.ts2 and Relationship0.te1 ≤ Relationship.te2) or (Relationship0.ts1 ≤ Relationship0.ts2 and Relationship.te1 ≥ Relationship.te2) RETURN Ts1,Te1 CREATE (Mariano_Pavone: Mariano_Pavone)-[Relationship: isTeammate {type:'property',rdft_hasStartTime:Ts1,rdft_hasEndTime:Te1,rdft_hasNumUpdate:1}]-(Cruz_Azul: Cruz_Azul)

(2) SPARQL[t] querying results of Q_1-Q_9 on the YAGO dataset

Queries	Querying results
Q_1	"1977-09-23" "Mestre"
Q_2	"2001-02-03" "2002-10-09" "Cultural_y_Deportiva_Leonesa" "2005-03-23" "2006-06-17" "CD_Guijuelo" "2006-12-01" "2007-11-14" "CD_Tudelano" …
Q_3	Return the first 15 records of Ricardo_Souza_Silva's personal information, such as: "Ricardo_Souza_Silva" wasBornIn "1963-02-17" 1 "São_Paulo" "Ricardo_Souza_Silva" playsFor "1982-10-04" "1985-01-28" 1 "Guaratinguetá_Futebol" "Ricardo_Souza_Silva" playsFor "1985-11-08" "1988-05-23" 2 "Nagoya_Grampus"
Q_4	Some of the data are as follows: "Scotland_national_under-19_football_team" "Scotland_national_under-21_football_team" "Falkirk_FC" "St_Johnstone_FC"
Q_5	Including the sex of the athlete and the time of birth: "Jonathan_Quick" "Milford_Connecticut" "Franco_De_Piccoli" "Tenirberdi_Suiunbaev" "Ion_Oncescu" "Bucharest"
Q_6	"Club_Olimpo" "Club_Atlético_River_Plate" "Club_de_Gimnasia_y_Esgrima_La_Plata" "LDU_Quito"
Q_7	TRUE/FALSE
Q_8	The results are stored in Neo4J database
Q_9	All triple sets with ?People as the subject

Moreover, we implemented a prototype system to support RDFt temporal data representation and querying, and the case studies and experimental results on two datasets of NBA players and YAGO verify the feasibility of the proposed approach.

VII. CONCLUSIONS AND FUTURE WORK

In order to represent temporal data with both the time information and the update count information, in this paper we proposed a new temporal RDF data representation model RDFt and its corresponding querying mechanism. We defined the syntax and semantics of the RDFt model in detail. Further, we proposed a corresponding query language of the RDFt model called SPARQL[t] by extending the standard RDF query language SPARQL. We presented the query syntax and operations of SPARQL[t]. Moreover, a querying transformation algorithm from SPARQL[t] to SPARQL was proposed, in order to achieve compatibility with the existing RDF query engines. Finally, we implemented a prototype system that can support RDFt temporal data representation and querying, and case studies and experimental results on two datasets of NBA players and YAGO verify the feasibility of the proposed approach.

In future works, we aim at testing our approach and the performance of prototype with more cases, and considering and investigating more temporal features based on RDF.

APPENDIX 1

See Table 17.

APPENDIX 2

See Fig. 12.

APPENDIX 3

See Fig. 13.

APPENDIX 4

See Table 18.

APPENDIX 5

See Table 19.

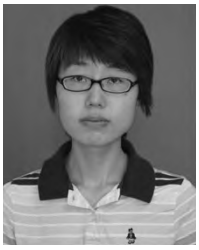
ACKNOWLEDGMENTS

The authors thank the anonymous referees for their valuable comments and suggestions, which improved the technical content and the presentation of the paper.

REFERENCES

[1] (2014). *RDF 1.1 Primer, W3C Working Group*. [Online]. Available: http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/
[2] A. Sözer, A. Yazici, H. Oguztüzün, and O. Taş, "Modeling and querying fuzzy spatiotemporal databases," *Inf. Sci.*, vol. 178, no. 19, pp. 3665–3682, Oct. 2008.
[3] D. Ognyanov and A. Kiryakov, "Tracking changes in RDF (S) repositories," in *Proc. Int. Conf. Knowl. Eng. Knowl. Manage.*, Oct. 2002, pp. 373–378.
[4] M. Gergatsoulis and P. Lilis, "Multidimensional RDF," in *Proc. OTM Confederated Int. Conf. Move Meaningful Internet Syst.*, Oct. 2005, pp. 1188–1205.
[5] C. Gutierrez, C. Hurtado, and A. Vaisman, "Temporal RDF," in *Proc. Eur. Semantic Web Conf.*, May 2005, pp. 93–107.
[6] A. Pugliese, O. Udrea, and V. S. Subrahmanian, "Scaling RDF with time," in *Proc. 17th Int. Conf. World Wide Web*, Apr. 2008, pp. 605–614.
[7] B. Motik, "Representing and querying validity time in RDF and OWL: A logic-based approach," in *Proc. Int. Semantic Web Conf.*, Apr. 2012, pp. 3–21.
[8] N. Noy, (2014). *Rector A Defining N-Ary Relations on The Semantic Web. W3C Working Group Note*. [Online]. Available: http://www.w3.org/TR/swbp-n-aryRelations/
[9] O. Udrea, D. R. Recupero, and V. S. Subrahmanian, "Annotated RDF," *ACM Trans. Comput. Log.*, vol. 11, no. 2, p. 10, 2010.
[10] Y. Wang, M. Zhu, L. Qu, M. Spaniol, and G. Weikum, "Timely YAGO: Harvesting, querying, and visualizing temporal knowledge from Wikipedia," in *Proc. 13th Int. Conf. Extending Database Technol.*, Mar. 2010, pp. 697–700.
[11] F. Grandi, "Multi-temporal RDF ontology versioning," in *Proc. 3rd Int. Workshop Ontology Dyn.*, Oct. 2009, pp. 1–10.
[12] M. Dylla, M. Sozio, and M. Theobald, "Resolving temporal conflicts in inconsistent RDF knowledge bases," *Coordination Chem. Rev.*, vol. 2, no. 1, pp. 474–493, 2011.
[13] J. Huber, "Temporal reasoning for RDF (S): A Markov logic based approach," *Arbeitspapier*, vol. 2, May 2014, pp. 1–134.
[14] A. Zimmermann, N. Lopes, A. Polleres, and U. Straccia, "A general framework for representing, reasoning and querying with annotated Semantic Web data," *Web Semantics, Sci., Services Agents World Wide Web*, vol. 11, no. 3, pp. 72–95, Mar. 2012.
[15] (2014). *RDF Schema 1.1. W3C Recommendation*. [Online]. Available: https://www.w3.org/TR/rdf-schema/
[16] (2014). *SPARQL 1.1 Query Language, W3C Recommendation*. [Online]. Available: https://www.w3.org/TR/sparql11-query
[17] B. McBride, and M. Butler, "Representing and querying historical information in RDF with application to E-discovery," in *Proc. ISWC*, Oct. 2009, pp. 1–13.
[18] (2019). *Neu4J*. [Online]. Available: https://Neo4J.com/
[19] C. Gutierrez, C. A. Hurtado, and A. Vaisman, "Introducing time into RDF," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 2, pp. 207–218, Feb. 2007.
[20] Z. Ma, M. A. M. Capretz, and L. Yan, "Storing massive resource description framework (RDF) data: A survey," *Knowl. Eng. Rev.*, vol. 31, no. 4, pp. 391–413, Sep. 2016.
[21] (2019). *NBA Data Website*. [Online]. Available: http://www.stat-nba.com/playerList.php
[22] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum, "YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia," *Artif. Intell.*, vol. 194, pp. 28–61, Jan. 2013.
[23] K. Bereta, P. Smeros, and M. Koubarakis, "Representation and querying of valid time of triples in linked geospatial data," in *Proc. Extended Semantic Web Conf.*, May 2013, pp. 259–274.
[24] A. Rodriguez, R. McGrath, Y. Liu, and J. Myers, "Semantic management of streaming data," in *Proc. 2nd Int. Conf. Semantic Sensor Netw.*, Oct. 2009, pp. 80–95.
[25] A. Rula, M. Palmonari, A. Harth, S. Stadtmüller, and A. Maurino, "On the diversity and availability of temporal information in linked open data," in *Proc. Int. Semantic Web Conf.*, Nov. 2012, pp. 492–507.
[26] S. Bykau, J. Mylopoulos, F. Rizzolo, and Y. Velegrakis, "On modeling and querying concept evolution," *J. Data Semantics*, vol. 1, pp. 31–55, May 2012.
[27] GaoS, GuJ, ZanioloC, "RDF-TX: A fast, user-friendly system for querying the history of RDF knowledge bases," in *Proc. 19th Int. Conf. Extending Database Technol. (EDBT)*, Jul. 2016, pp. 269–280.
[28] C. Zaniolo, S. Gao, M. Atzori, M. Chen, and J. Gu, "User-friendly temporal queries on historical knowledge bases," *Inf. Comput.*, vol. 259, no. 3, pp. 444–459, Apr. 2018.
[29] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic, "EP-SPARQL: A unified language for event processing and stream reasoning," in *Proc. 20th Int. Conf. World Wide Web*, Mar. 2011, pp. 635–644.
[30] D. Le-Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth, "A native and adaptive approach for unified processing of linked streams and linked data," in *Proc. Int. Semantic Web Conf.*, Oct. 2011, pp. 370–388.
[31] D. F. Barbieri, D. Braga, S. Ceri, and E. D. Valle, "Querying RDF streams with C-SPARQL," *ACM SIGMOD Rec.*, vol. 39, no. 1, pp. 20–26, Sep. 2010.

FU ZHANG received the Ph.D. degree from Northeastern University, China, in 2011, where he is currently an Associate Professor and a Ph.D. Supervisor with the School of Computer Science and Engineering. He has authored more than 40 refereed international journals and conference papers. His research work was published in high-quality international conferences, such as CIKM and DEXA, and in highly cited international journals, such as *Fuzzy Sets and Systems*, *Knowledge-Based Systems*, and *Integrated Computer-Aided Engineering*. He has also authored two monographs published by Springer. His current research interests include RDF data management, ontology, knowledge graph, and knowledge representation and reasoning.



KE WANG is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Northeastern University, China. Her current research interests include RDF data management, spatial and temporal data management, and ontology.

ZHIYIN LI is currently pursuing the master's degree with the School of Computer Science and Engineering, Northeastern University, China. Her current research interests include RDF data management, and spatial and temporal data management.

JINGWEI CHENG received the Ph.D. degree from Northeastern University, China, in 2011, where he is currently with the School of Computer Science and Engineering. He has authored more than 20 refereed international journals and conference papers (e.g., WI and DEXA). He has also authored one monograph published by Springer. His current research interests include description logics, RDF data management, and ontology.