Temporality in Semantic Web

Di Wu

2017
## Contents

1 Introduction ................................................. 5

2 Semantic Web Concepts, Languages, and Implementations .......... 7
   2.1 Ontology ............................................... 7
   2.2 Description Logic ................................... 10
   2.3 RDF: Resource Description Framework ...................... 12
      2.3.1 RDFS ............................................. 15
      2.3.2 RDFa ............................................. 16
      2.3.3 Turtle ............................................ 17
      2.3.4 N Triple ....................................... 18
      2.3.5 N3 - Notation 3 .................................. 18
      2.3.6 Reifications .................................... 19
   2.4 OWL: Web Ontology Language .................................. 20
   2.5 Linked Data Initiative ................................... 23
      2.5.1 DCMI: Dublin Core Metadata Initiative ................. 24
      2.5.2 FOAF: Friend of a Friend ........................ 25
      2.5.3 Google Knowledge Graph .......................... 27
      2.5.4 Schema.org ..................................... 28
      2.5.5 Linked Open Vocabularies .......................... 28

3 Time Ontology ................................................ 30
   3.1 Time Ontology Review ................................... 30
      3.1.1 Temporal Features ................................ 31
      3.1.2 Temporal Elements and Structures .................. 33
      3.1.3 Temporal Properties ................................ 34
      3.1.4 Temporal Relations ................................ 35
   3.2 Time Ontology Development ................................ 36
      3.2.1 Temporal Relations ................................ 36
      3.2.2 Linking Time and Events ........................... 37
      3.2.3 Measuring Durations ................................ 37
      3.2.4 Clock and Calendar ................................ 37
      3.2.5 Describing Times and Durations .................... 38
   3.3 Time Ontology in OWL .................................... 38

4 Temporal Database ............................................ 39
   4.1 Concept of Temporal Database ................................ 40
      4.1.1 Historical Database ................................ 41
Abstract

The Semantic Web concept has been formally created two decades ago [1, 2]; and its implementations such as RDF (Resource Description Framework) are developed and widely used [12]. The Semantic Web and its implementations largely improve the usage of separate data on the Internet thus make more data available for queries. However, nowadays, along with the Information Explosion and Data Flooding, the data on the Internet are too much rather than too few. Not only a lot of data are redundant but also part of the data are out of date. Thus, the old problem "Finding more data" transits to "Finding accurate data". We believe that Temporality in Semantic Web can help to solve this problem. In this survey paper, we explore both current Semantic Web techniques and implementations and Temporal database concepts to find a path that how these two fields can be integrated together to satisfy current and future needs.
1 Introduction

This section will briefly review the concept of Semantic Web by discussing its creation, current achievements, and future challenges.

Although the concept "Semantic Web" has been created as early as 1960s [1, 2, 3, 4, 5], the best formal and widely accepted definition is the one in the 2001 article - "The Semantic Web" - by Tim Berners-Lee et al, that is "an extension of the current one[WWW], in which information is given well-defined meaning, better enabling computers and people to work in cooperation" [6]. In the article, the authors point out that the motivation for a Semantic Web is based on the increasing need for information-exchange among machine agents and the inability of current WWW to satisfy this goal. Semantic web aims at the meaning of information, thus, it has most expressive rules for knowledge representation. Important techniques in the field are also developed: eXtensible Markup Language(XML) has tags embedded that allow people to assign meaning for certain information; Resource Description Framework (RDF) uses a set of triples to express meaning, and turns out to the recommend standard of the Semantic Web by W3C; Universal Resource Identifier (URI) provides a link to every subject, object, and relationship thus makes linkage and integration among different knowledge bases possible [6].

W3C led the specifying, developing and deploying languages for sharing meaning for semantic web. Besides RDF and URI, Web Ontology Language (OWL) uses links of RDF to allow ontologies across systems [7]. Bernstein et al in their 2016 review for the semantic web [45] states that:

1) Over 2.5 billion web pages have markup conforming to the schema.org format;
2) Linked data is widely adopted by major libraries and museums as well as media sites such as BBC and New York Times;
3) Web companies are developing knowledge graphs that link massive number of entities together such as Google, Bing, and Facebook;
4) Commercial database systems are providing native support for Semantic Web languages such as Oracle;
5) Recommender companies are taking advantage of semantics and tagging to improve their accuracy;
6) The world health organization is developing the main international terminology for diseases as an ontology for semantic web users; etc.
However, As Lee et al [7] pointed out, the development is still below the expectations of the founders of the Semantic Web. Not only that we still have few means to easily generate semantic web annotations but also it is challenging that how to effectively query huge number of decentralized information repositories of varying scales [7]. In addition, Bernstein et al [45] point out that the research of Semantic Web has transitioned into larger, more applied systems. It builds on earlier foundations but now has a diverse set of pursuits. For example, the rapid development of smart phones and mobile devices, leads to the new generation of applications such as mobile healthcare and mapping. These applications have a different environment than the old days formal and precise queries that were expected by former founders of Semantic Web.

In addition to the current problems of the Semantic Web that we discussed above, we include one more as a future challenge in this survey paper, which is the temporality of the Semantic Web. Although current Semantic Web implementations support some temporal elements, however, it is not enough to solve the problem "Finding accurate current data". Besides we include temporal elements in the semantics, we also need to attach some kind of timestamp to statements that compose the database so that we can compare and filter the most up-to-date data. We will discuss this problem in more detail in the last section.

This survey paper first briefly reviews the concepts of the Semantic Web. We then discuss the current techniques and implementations of the Semantic Web, such as RDF and its annotations, OWL, and Linked Data Initiative and its projects. Then, we examine the Time Ontology as a psychological nature. Next, we discuss important concepts in Temporal Database such as different type of databases, and Temporal Relational Algebra and Calculus. The SQL2011 extension for temporality is discussed as an example as well. After that, how current Semantic Web techniques and implementations support temporality is discussed. At last, we conclude this paper with possible connections between these two fields and future research directions.
2 Semantic Web Concepts, Languages, and Implementations

As a response to the call for sharing the meaning for semantic web, standards organizations such as IETF and W3C led the specifying, developing an deploying languages that creates the foundation for data integration and information sharing [7]. The underlying concepts such as Ontology, Description Logic, and Linked Data Initiative, the languages such as RDF, and OWL, and the implementations such as Google Knowledge Graph are discussed in this section.

2.1 Ontology

An ontology is an explicit specification of a conceptualization - the terms in the domain and the relationship among them [8]. It defines a commonly accepted vocabulary for a domain for information sharing - not only human beings but also machine agents [9]. A knowledge base is created by an ontology that defines some classes / concepts together with the instances of these classes [9]. Developing an ontology, includes defining classes and their hierarchy, as well as the properties that describing the concept with restrictions in possible values, and instances in these classes [9]. Although developing an ontology is a creative process and there are many ways to do so, because the purpose of ontologies is to share information for human and machines, there are some guidelines for creating classes, ordering the hierarchy, and naming, etc [9]. The main requirements for a good ontology include well-defined syntax, efficient support for reasoning, a clear and formal semantics, sufficient power of expression, and easy to express [10].

We follow Noy and McGuinness, "Ontology development 101: A guide to creating your first ontology"[9] to discuss the detailed steps of developing an ontology as below.

Step1: Domain and Scope. Defining the domain and scope of an ontology is the first and important step of developing an ontology. We need to define the object that the ontology is going to cover and describe. Also, we need to have sufficient but not abundant information to provide answers for possible questions. Further more, we need to identify users groups that will use the ontology.

For example, we are going to build a database for recording information of students in the Graduate Center, CUNY. Then the ontology of student need to cover all types of students and their information. Users of this ontology and database include the database
managers, as well as faculties and stuffs in CUNY, and students themselves. Thus, restrictions on certain attributes need to be implemented. The database managers, will design and maintain the ontology. In conclusion, the student ontology will represent the group of students, including their personal information, registration and course information, billing and financial information, etc.

Step 2: Reuse existing ontologies. There are many advantages of reusing existing ontologies. At first, it is more efficient to build an ontology by refining and extending existing ones than start from scratch. Second, many times the ontology needs to interact with other applications or systems, thus, a common foundation can make this interaction smoother. The last, it is not difficult to import/export/translate from existing ontologies.

In our ontology of student, since the topic student is very common and generic, it is not hard to find an existing ontology that has been tested and used widely. Thus, instead of creating a new one, we can reuse those and make little bit modification for our special needs.

Step 3: Defining important terms. Important terms are the ones that will be used for statements or explanations in the system.

Definition of terms helps users to understand the ontology and thus avoid errors. For example, registration status, is defined to be the status of a selected term of a current student's registration. The possible values of this terms are: part-time - who registered less than 9 credits; full-time - who registered 9 credits or more; absent - who registered 0 credit, etc. Another term, student status, is defined to represent the state of of the relationship between the student and the GC: past/alumni means the student is graduated; current means the student is studying; and future means the student is admitted and will attend in the future.

Step 4: Defining classes and class hierarchy. Based on similarity and difference, classes are to category and group terms together. Since classes have the properties that are shared by all instances of the class and the subclasses, hierarchy of classes is to provide a structure from the most general concept to most specific subconcepts.

In the ontology of Student, as Student is the top level, and we can include the subclasses of the student by the level of study such as Undergraduate, Master, and PhD.
Or, we can make subclass based on the major of the student such as Physics, Computer Science, Math, etc. If we take the second approach, the hierarchy of class is Student, Computer Science, PhD.

Step5: Defining properties of concepts. As Noy and McGuinness pointed out, Step4 and 5 are conducted together. Defining classes actually involves defining the properties or attributes of the classes. Since subclasses inherit all properties or attributes from their superclass, properties need to be defined along with the hierarchy of classes too. The super class has a general properties that applies to its subclasses while the subclass has a specific property that is different than other subclasses.

Continue with our Student ontology. The properties of the Student class include profile and personal information such as name, address, email, phone, etc, and this properties will be inherited by the subclasses of Student. The class PhD may has properties such as level, mentor, study area, etc.

Step6: Defining facets of properties. For each property of a class, we need to define the facets to describe it, such as type of the value, allowed/disallowed value, cardinality of the value, and other features of the value.

For example, the property phone of class Student might be limited to 3 sets of numbers, the 1st is the country code, and 2nd is the area code, the 3rd is the number. The property study area might be a list of pre-defined areas as allowed values such as Algorithms, Data Science, Artificial Intelligence, Machine Learning, etc. We can also define the facets of this property as open to the user so any value and be taken.

Step7: All previous 6 steps actually create a structure of a system; and the last step is to fill value for the properties and create a instance. Every instance has an associated class in the hierarchy and has some value of the properties. This step is a transition from an abstract concept to a specified object.

In the last step, we can create William Weber as an instance of the PhD student in Computer Science in the GC. To do that, we need to fill the most general information such as personal information per the requirement of Student class, and most specific information such as mentor per the PhD class requirements. The 1-6 steps define an abstract structure of the Student, and last step creates a real instance.
2.2 Description Logic

Description Logic is widely used in ontological modeling. Description Logic Ontologies consist of a set of statements, which are called axioms that must be true. Description Logic has several well known syntactic variants, and following figure shows some conventional notations.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Example</th>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\top$</td>
<td>$\top$ is a special concept with every individual as an instance</td>
<td>$\top$</td>
<td>top</td>
</tr>
<tr>
<td>$\bot$</td>
<td>empty concept</td>
<td>$\bot$</td>
<td>bottom</td>
</tr>
<tr>
<td>$\sqcap$</td>
<td>intersection or conjunction of concepts</td>
<td>$C \sqcap D$</td>
<td>C and D</td>
</tr>
<tr>
<td>$\sqcup$</td>
<td>union or disjunction of concepts</td>
<td>$C \sqcup D$</td>
<td>C or D</td>
</tr>
<tr>
<td>$\neg$</td>
<td>negation or complement of concepts</td>
<td>$\neg C$</td>
<td>not C</td>
</tr>
<tr>
<td>$\forall$</td>
<td>universal restriction</td>
<td>$\forall R. C$</td>
<td>all R-successors are in C</td>
</tr>
<tr>
<td>$\exists$</td>
<td>existential restriction</td>
<td>$\exists R. C$</td>
<td>an R-successor exists in C</td>
</tr>
<tr>
<td>$\sqsubseteq$</td>
<td>Concept inclusion</td>
<td>$C \sqsubseteq D$</td>
<td>all C are D</td>
</tr>
<tr>
<td>$\equiv$</td>
<td>Concept equivalence</td>
<td>$C \equiv D$</td>
<td>C is equivalent to D</td>
</tr>
<tr>
<td>$\equiv$</td>
<td>Concept definition</td>
<td>$C \equiv D$</td>
<td>C is defined to be equal to D</td>
</tr>
<tr>
<td>$:$</td>
<td>Concept assertion</td>
<td>$a : C$</td>
<td>a is a C</td>
</tr>
<tr>
<td>$:$</td>
<td>Role assertion</td>
<td>$(a, b) : R$</td>
<td>a is R-related to b</td>
</tr>
</tbody>
</table>

Figure 1: Description Logic Notations

For convenience, these statements are grouped into three categories: assertional (ABox) axioms that record the attributes/information about an object; terminological (TBox) axioms that describes relationship among concepts; and relational (RBox) axioms that refer to the properties of roles(inclusion, equivalence, and disjointness) [11].
ABox axioms represent attributes or facts of the object, such as concept, role, individual inequality and individual equality. For example,

\[
William : \text{GraduateStudent}
\]  

(1)

is a concept assertion that describes William is a student. or, we can read as individual William is an instance of the concept Graduate Student.

\[(William, \text{CUNY}) : enrolled\]

(2)

is a role assertion that describes William is enrolled in CUNY.

\[
William \approx Bill
\]

(3)

is an individual equality assertion that describes William and Bill actually are describing the same individual. And

\[
William \neq Jack
\]

(4)

is an individual inequality assertion that describes William and Bill are different individuals.

TBox axioms represent relationships between concepts. For example,

\[
\text{GraduateStudent} \sqsubseteq \text{Student}
\]

(5)

describes the fact that all graduate students are students; and

\[
\text{Woman} \equiv \text{Female}
\]

(6)

describes the fact that the concept Woman is the same as the concept Female.

RBox axioms represent the property of roles. Most common RBox axioms are role inclusion and role equivalence axioms. For example,

\[
enrolledInQC \sqsubseteq enrolledInCUNY
\]

(7)

describes the fact that all students who enroll in the Queens College are also enroll in CUNY since Queens College is one of the senior colleges in CUNY system. Description logic uses these three basic types of axioms to construct complex concepts and to describe complex situations.
SROIQ is the resulting logic of extending the description logic underlying OWL-DL, SHOIN by adding complex role inclusion axioms, reflexive, antisymmetric, irreflexive, disjoint roles, universal role, etc [11].

2.3 RDF: Resource Description Framework

Traditional Relational Database represents information either by Column (for specific topics) or by Row (for some entities). RDF makes information represented by Cell (a piece of information) possible. The first public draft about RDF was released in 1997 by W3C as a foundation for processing metadata [12]. The RDF data model is based on a set of statements, which are subject-predicate-object triples. The subject in the triple represents a resource, and the predicate represents the relationship between the subject and the object. The object can be regarded as attributes or properties of the resource. There are various ways to encode the triple, however, Universal Resource Identifier (URI) is the most major component used for identifying resources and recommended by the W3C Semantic Web activity. Along with the development of RDF, RDF Schema (Resource Description Framework Schema) is created as an extension of RDF. RDFS is a set of classes with some characteristics and can be displayed using RDF extensible knowledge representation data model. Many RDFS components are included in Web Ontology Language (OWL) which will be discussed later on. Also, RDF vocabularies such as FOAF and Dublin Core will be discussed as well.

The following part of this section follows mainly based on W3C RDF 1.1 XML Syntax which is W3C recommendation Feb 25, 2014 [18].

Let’s look at a simple example as the statement: William Weber lives in NYC. This triple has the individual, William Weber, as the subject; has predicate subscribing the "lives in" relationship; and object is the city of New York. In terms of graph, we can state the expression as below:

This graph is incomplete in RDF view since it lacks of URIs that identify the resources. Thus, we need to rebuild the graph as below to represent the statement more precisely:

In above graph, Literal "William Weber" is in a rectangle and concepts are filled with their URI in ovals. The predicate "lives in" is labeled with its Name Space Name.
RDF/XML language can translate from the graph to code from step by step. rdf: Description is for node element; and ex: livesIn and ex: fullName are the property elements.

```
<rdf:Description>
  <ex:livesIn>
    <rdf:Description>
    </rdf:Description>
  </ex:livesIn>
</rdf:Description>
```

We need to embed URI of the resources if they have, thus, resulting RDF/XML code is as below:

```
<a xlink:href="http://www.example.org/New York City">New York City</a>
```

Figure 2: Triple statement

Figure 3: Triple statement in RDF
Note that both ex:livesIn and ex:fullName are attributes of the same node. Thus, we can make the code simpler by using multiple property elements on a node element as below:

```xml
<rdf:Description rdf:about="http://www.example.org/WWeber">
  <ex:livesIn>
    <rdf:Description rdf:about="http://www.example.org/NewYorkCity">
    </rdf:Description>
  </ex:livesIn>
  <ex:fullName>
    <rdf:Description>
      William Weber
    </rdf:Description>
  </ex:fullName>
</rdf:Description>
```

Note that there are an empty property elements and an element’s content is string literal. We can further abbreviate the code as below:

```xml
<rdf:Description rdf:about="http://www.example.org/WWeber">
  <ex:livesIn rdf:resouces="http://www.example.org/NewYorkCity">
    William Weber
  </ex:livesIn>
</rdf:Description>
```
Then the last part is to add RDF frame so that machines can know what syntax the code is following.

```xml
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex="http://example.org/">
    <rdf:Description rdf:about="http://www.example.org/WWeber">
        <ex:livesIn rdf:resources="http://www.example.org/NewYorkCity"></ex:livesIn>
        <ex:fullName>William Weber</ex:fullName>
    </rdf:Description>
</rdf:RDF>
```

We can conclude with a more pseudo code style to represent a RDF triple, Subject, predicate(s), and Object, as below:

```xml
<rdf:Description rdf:about="Subject">
    <Predicate rdf:resources="Object"/>
    <Predicate>Literal</Predicate>
</rdf:Description>
```

2.3.1 RDFS

RDF Schema extends the basic RDF vocabulary and provides a data-modeling vocabulary for RDF data. This section is based on W3C's latest recommendation 25 Feb 2014.

As we discussed in previous section, RDF can describe a triple, and lacks of the efficiency to express complicated objects. RDF Schema adds the ability of describing groups of related resources and the relationships among these resources. It follows the logic of Object-oriented programming languages such as Java and defines classes and associated properties. However, RDFS focuses on properties instead of objects. RDFS describes properties in terms of the classes of resources to which they apply, rather than describes classes based on similarity of properties of their instances.

Two major categories of the RDFS syntax are Classes and Properties. Class includes Resource, Class, Literal, Datatype, langString, HTML, XMLLiteral, Property etc as subclasses, and Properties includes range, domain, type, subClassOf, subPropertyOf, label, comment etc. RDFS also has additional classes and properties such as Container
Classes and Properties, RDF Collections, Reification Vocabulary, Utility Properties. The detailed list of RDFS classes and properties are shown as below:

<table>
<thead>
<tr>
<th>Class name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:Resource</td>
<td>The class resource, everything.</td>
</tr>
<tr>
<td>rdfs:Literal</td>
<td>The class of literal values, e.g. textual strings and integers.</td>
</tr>
<tr>
<td>rdf:langString</td>
<td>The class of language-tagged string literal values.</td>
</tr>
<tr>
<td>rdf:HTML</td>
<td>The class of HTML literal values.</td>
</tr>
<tr>
<td>rdf:XMLLiteral</td>
<td>The class of XML literal values.</td>
</tr>
<tr>
<td>rdfs:Class</td>
<td>The class of classes.</td>
</tr>
<tr>
<td>rdfs:Property</td>
<td>The class of RDF properties.</td>
</tr>
<tr>
<td>rdfs:Datatype</td>
<td>The class of RDF datatypes.</td>
</tr>
<tr>
<td>rdf:Statement</td>
<td>The class of RDF statements.</td>
</tr>
<tr>
<td>rdf:Bag</td>
<td>The class of unordered containers.</td>
</tr>
<tr>
<td>rdf:Seq</td>
<td>The class of ordered containers.</td>
</tr>
<tr>
<td>rdf:Alt</td>
<td>The class of containers of alternatives.</td>
</tr>
<tr>
<td>rdfs:Container</td>
<td>The class of RDF containers.</td>
</tr>
<tr>
<td>rdfs:ContainerMembershipProperty</td>
<td>The class of container membership properties, rdfs:_1, rdfs:_2, ..., all of which are sub-properties of 'member'.</td>
</tr>
<tr>
<td>rdf:List</td>
<td>The class of RDF Lists.</td>
</tr>
</tbody>
</table>

2.3.2 RDFa

RDFa (Resource Description Framework in Attributes) is a W3C Recommendation that adds a set of attribute level extensions to HTML and various XML-based document types for embedding metadata. Current status is the RDFa 1.1 in June 2012 as a recommendation by W3C. RDFa 1.1 no longer relies on XML-specific namespace mechanism thus non-XML document types such as HTMLs can use RDFa 1.1 as well. The essence of RDFa is to set up a set of attributes that can be used for embedding metadata in a website document. Each attributes actually is a simple URL which will explain the meaning of the data. For example, the above statement "William Weber lives in New York City" can be simply expressed in RDFa as below:

```html
```
2.3.3 Turtle

A turtle document enables representing a RDF graph in a compact textual format. W3C provides an example turtle document describes the relationship between Green Goblin and Spiderman:

```turtle
@base <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rel: <http://www.perceive.net/schemas/relationship/> .

<#green-goblin>
    rel:enemyOf <#spiderman> ;
    a foaf:Person ; # in the context of the Marvel universe
    foaf:name "Green Goblin" .

<#spiderman>
    rel:enemyOf <#green-goblin> ;
    a foaf:Person ;
```
Note that beside the namespace definitions as we have seen in RDF, turtle is very compact because of the use of column to list predicates, and comma to list objects.

### 2.3.4 N Triple

N-Triples corresponds directly to the raw RDF triples and uses fully unabbreviated URIs. The subject, predicate and object of an RDF Triple are represented in a sequence and separated by white space. This sequence is terminated by a ‘.’ and a new line. W3C provides a simple example of N Triple as:

```xml
<http://one.example/subject1> <http://one.example/predicate1>
<http://one.example/object1> .
```

### 2.3.5 N3 - Notation 3

Notation 3 (N3) is a more compact serialization of RDF. Namespaces are used to abbreviate the statements. Thus, N3 provides a compact representation. Similar to Turtle, ‘;’ is used to indicate another triple with the same subject, and ‘,’ is used to separate objects.

For example, the RDF model in XML notation:

```xml
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about="http://en.wikipedia.org/wiki/Tony_Benn">
        <dc:title>Tony Benn</dc:title>
        <dc:publisher>Wikipedia</dc:publisher>
    </rdf:Description>
</rdf:RDF>
```

can be written in Notation 3 like this:

```turtle
@prefix dc: <http://purl.org/dc/elements/1.1/>.
```
2.3.6 Reifications

As we already discussed, RDF and OWL make statements using triples. Each statement in logic is a binary relationship. Reification, is using a built-in vocabulary in RDF for describing statements so that we know more than the merely binary relationship, or, in other words, is making statements about other statements. In practice, Reification always uses four statements for describing statements, and thus is called "reification quad" [17].

Since each statement is a triple about resources. To make a statement about other statements, we first need to make the statement to be described as a resource so that we can refer to.

```
<rdf:Description rdf:about="949352">
  <uni:name>William Weber</uni:name>
</rdf:Description>
```

Then we can reify the statement as:

```
<rdf:Statement rdf:about="StatementAbout949352">
  <rdf:subject rdf:resource="949352"/>
  <rdf:predicate rdf:resource="&uni;name"/>
  <rdf:object>William Weber</rdf:object>
</rdf:Statement>
```

Note that the rdf:subject, rdf:predicate, and rdf:object are used so we can access the triple of the statement.

However, this standard reification has been criticized for its lack of efficiency. While reification itself is simple and intuitive, the standard reification does not have formal semantics. As a result, researchers have developed several other reifying methods to overcome the inefficiency issue. One approach is Singleton Property, that instead of
making statement for the other statement to describe some additional property, Singleton property approach creates new property that embeds both the original property and the additional property. Singleton properties treats multiple properties at the same time so they are also called meta knowledge assertions. Following Nguyen et al’s paper "Don’t like RDF Reification?" [46], the example is:

Another approach is N-ary relations. The N-ary relations approach creates an intermediate resource to denote the relationship. Instead of having the subject in RDF triple being a single value, the N-ary relations states that the subject is involved in a relationship that has not only a value but also some extra properties.

The last approach to be included is named graphs. Named graphs extends the RDF triple relationship and includes additional nodes. The RDF triple and those additional nodes together as a set of quadruples represent the reification as a whole.

Reifications are used to express temporal elements in practice, we will discuss more about this topic later.

### 2.4 OWL: Web Ontology Language

The W3C Web Ontology Language is also a Semantic Web language. It is designed to express rich information. OWL is a logic-based language, and machine agents are able to exploit knowledge expressed in OWL. OWL 2 as the most current version was released in 2012 by W3C recommendation. The OWL 2 ontology provides structures as
classes, properties, individuals, and data values, and are stored as Semantic Web documents. OWL 2 works well with RDF; and can be converted into RDF graph, and back versa. RDF/XML is the only mandatory syntax of OWL 2, and other concrete syntaxes may be used for compliment or specific purpose, such as OWL/XML, Functional Syntax, Manchester Syntax, and Turtle, etc [15]. For simplicity purpose, we use RDF/XML to represent OWL in examples.

Back to our example we used before, that William Weber is a graduate student. At this point, what we know is, William Weber is an instance of class "GraduateStudent", and class "GraduateStudent" should be a subclass of "Student" which may have many more subclasses such as "UndergraduateStudent", "HighSchoolStudent", etc. How can we define in OWL and express this example nicely?

We first prepare the OWL header part as below:

```xml
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:dc="http://purl.org/dc/elements/1.1/">
   <!-- OWL Header Example -->
   <owl:Ontology rdf:about="http://www.example.org/student">
     <dc:title>Example Student Ontology</dc:title>
     <dc:description>An example ontology written for RDFS & OWL introduction tutorial</dc:description>
   </owl:Ontology>
</rdf:RDF>
```

Then we can extend the heading and include the class definition as below. For brevity, the heading part is omitted.

```xml
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:dc="http://purl.org/dc/elements/1.1/">
   <!-- OWL Header Omitted -->
   <!-- OWL Class Definition - Student Level-->
```

21
We then can continue to define two subclasses of the "Student": the "GraduateStudent" and "UndergraduateStudent" as below:

```xml
<owl:Class rdf:about="http://www.example.org/student#level">
  <rdfs:label>Student Level</rdfs:label>
  <rdfs:comment>The class of all student levels.</rdfs:comment>
</owl:Class>

<owl:Class rdf:about="http://www.example.org/student#graduate">
  <rdfs:subClassOf rdf:resource="http://www.example.org/student#level"/>
  <rdfs:label>Graduate Student</rdfs:label>
  <rdfs:comment>Graduate Student</rdfs:comment>
</owl:Class>

<owl:Class rdf:about="http://www.example.org/student#undergraduate">
  <rdfs:subClassOf rdf:resource="http://www.example.org/student#level"/>
  <rdfs:label>Undergraduate Student</rdfs:label>
  <rdfs:comment>Undergraduate Student</rdfs:comment>
</owl:Class>
```
After we defined the structure/hierarchy of classes, we can add William Weber as an instance of the "Graduate Student" class.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:students="http://www.example.org/student#">
  <!-- OWL Header Omitted -->
  <!-- OWL Class/subclass Definition Omitted -->
  <!-- Define William Weber as an instance of Graduate Student -->
  <rdf:Description rdf:about="http://www.example.org/WWeber">
    <!--William Weber is an individual (instance) of the graduate student class -->
    <rdf:type
      rdf:resource="http://www.example.org/student#graduate"/>
  </rdf:Description>
</rdf:RDF>
```

There are many things we can add into the classes, such as data type property and object property etc. However, for simplicity, we will not cover that much here.

### 2.5 Linked Data Initiative

Linked Data Initiative is part of the Semantic Web in the way that resources over the web of hypertext can be linked so human and machine agents can reach out to more data if they have some [13]. By the Linked Data, one of the important goal of the Semantic Web - reuse of data - is satisfied.

The four principles of linked data as Tim Berners-Lee proposed are [13]:
1) Use URIs as names for things;
2) Use HTTP URIs so that people can look up those names;
3) When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL);
4) Include links to other URIs so that they can discover more things.
W3C further developed Linked Open Data (LOD), which is Linked Data under an open licence.

A five star system is created to encourage people to create good linked data. In this system, the easier for people to use, the more powerful, more stars will be achieved:

1) you get 1 star if your data "available on the web (whatever format) but with an open licence, to be Open Data";
2) get 2 stars if your data "Available as machine-readable structured data (e.g. excel instead of image scan of a table)"
3) get 3 stars if your data "as (2) plus non-proprietary format (e.g. CSV instead of excel)"
4) get 4 stars if "All the above plus, Use open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your stuff"; and
5) get 5 stars if "All the above, plus: Link your data to other people's data to provide context" [13].

It is clearly, that open, structured, and linked data is preferred, which is align with the motivation of the Semantic Web.

Current databases quoted from wikipedia are:

### Datasets [ edit ]
- DBpedia – a dataset containing extracted data from Wikipedia; it contains about 3.4 million concepts described by 1 billion triples, including abstracts in 11 different languages
- FOAF – a dataset describing persons, their properties and relationships
- GeoNames provides RDF descriptions of more than 7,600,000 geographical features worldwide.
- UMBEL – a lightweight reference structure of 20,000 subject concept classes and their relationships derived from OpenCyc, which can act as binding classes to external data; also has links to 1.5 million named entities from DBpedia and YAGO
- Wikidata – a collaboratively-created linked dataset that acts as central storage for the structured data of its Wikimedia sister projects

#### 2.5.1 DCMI: Dublin Core Metadata Initiative

The Dublin Core Metadata Initiative is one of RDF vocabulary that has a metadata element and in the beginning it provides properties such as "creator", "publisher", and "title", etc [16]. This original 15 metadata elements are defined even before RDF so that they have a separate namespace. Nowadays DCMI is one of the most popular RDF vocabularies and it is inline with the Linked Data movement. Simple Dublin Core has 15...
elements while Qualified Dublin Core has 3 additional elements. The full list of elements can be found on DCMI’s website.

The Dublin Core Metadata Element Set - DCMI are:

- DC.TITLE
- DC.CREATOR
- DC.SUBJECT
- DC.DESCRIPTION
- DC.PUBLISHER
- DC.CONTRIBUTORS
- DC.DATE
- DC.TYPE
- DC.FORMAT
- DC.IDENTIFIER
- DC.SOURCE
- DC.LANGUAGE
- DC.RELATION
- DC.COVERAGE
- DC.RIGHTS

A simple pseudo code style for embedding DCMI in RDF/XML is shown as below:

```xml
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="http://www.example.org/thisPage">
    <dc:title>An example of DC implementation</dc:title>
    <dc:creator>William Weber</dc:creator>
    <dc:format>Web Pages</dc:format>
    <dc:identifier>ISBN 0123456789</dc:identifier>
  </rdf:Description>
</rdf:RDF>
```

2.5.2 FOAF: Friend of a Friend

As FOAF stands for Friend of a Friend, FOAF uses RDF technology to connect information on a person’s homepage with that of his/her friends, and the friends of his/her friends. Thus, FOAF enables machines to understand a page, and learn relations that
connect people, places, and things on the web [14]. Since most web pages are built by people, and covers some thing about people, FOAF can greatly expand the knowledge base of web as a whole. Also, FOAF is very easy to use and there are various FOAF tools for generating FOAF files such as FOAF site, foaf-a-matic, etc [14].

Main FOAF terms can be grouped as Core, Social Web, and Linked Data utilities. The core terms include Agent

- Person
- name
- title
- img
- depiction (depicts)
- familyName
- givenName
- knows
- based_near
- age
- made (maker)
- primaryTopic (primaryTopicOf)
- Project
- Organization
- Group
- member
- Document
- Image

A simple example of FOAF to describe a person might be (cite from foaf official website)

```xml
<foaf:Person rdf:about="#danbri"
            xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <foaf:name>Dan Brickley</foaf:name>
  <foaf:homepage rdf:resource="http://danbri.org/" />
  <foaf:openid rdf:resource="http://danbri.org/" />
  <foaf:img rdf:resource="/images/me.jpg" />
</foaf:Person>
```
2.5.3 Google Knowledge Graph

The Knowledge Graph is a knowledge base of Google to improve its search engine and enhance the result by adding a display of structured and detailed information besides the traditional list of links to other sites. The Knowledge Graph provides a convenient way for Google’s search engine users that they don’t need to go further for information if they are satisfied with the Knowledge Graph display. Also, Google Now exploits the result of the Knowledge Graph and uses the short summary for the keywords/topic as feedback to its users. The Knowledge Graph takes a semantic search from various sources include Wikidata and Wikipedia etc.
2.5.4 Schema.org

As we discussed above, there are many markups for the web, and webmasters are facing a hard situation: which one fits best, and have to learn it. The result is many of them either not have any markups, or they make incorrect ones. This result brings more trouble for data consumers such as search engines since they need to parse for improperly formed syntax and vocabulary. Many solutions are purposed, and we are going to discuss two of them here. This section will cover Schema.org which have achieved a big success in practice; and next section will cover Linked Open Vocabularies, which links different vocabularies together and provides an clear view of conversion.

Schema.org was created in 2011 by major search engines Bing, Google, and Yahoo. It aims at providing a single schema that covers a wide range of topics. While webmasters only need to with the markup once, search engines can use the markup differently. The idea of Schema.org actually align with RDF that a piece of information is represented as a triple by data provides and how to retrieve the information is the work of query developers.

Schema.org is very successful as it is expected. In the developer side, it grows from 297 classes and 187 relations when it is born to 638 classes and 965 relations in 2016. In the application side, a few applications across different companies have started to adopt Schema.org vocabulary. Google's Rich Snippets is the first one, Knowledge Graph and Gmail take advantage of Schema.org-based structured data markup too. In the webmasters side, the adoption of Schema.org is raised from 22 percent in 2015 to 31.3 percent in 2016 in a sample of 10 billion pages.

The success of Schema.org reveals that in order to make more people participant, we need to provide easy access for everyone to join. Recall the flow of usage of computer from professional technicians (before windows and WWW Internet) to personal computers, and then to key-board-free smart phones, along with the less difficulty of using the device, the population is extended significantly. As the Guha concluded at the end of the paper, three lessons must be learned from the failure of many vocabularies and the success of Schema.org as 1) easiness 2) recipes like documentation 3) start simple.

2.5.5 Linked Open Vocabularies

Besides Schema.org, another solution to deal with many vocabularies is Linked Open Vocabularies. Based on the view of vocabularies also data, the name Linked Open Vo-
vocabularies is derived from Linked Open Data. LOV provides a choice of many vocabularies based on a set of criteria such as URI stability and availability, proper version, etc. Most of the established vocabularies are collected in LOV, as well as their version history and relationships among each other as the example shows below. In short, LOV provides a tool for webmasters to choose best vocabulary with less considerations.
3 Time Ontology

Time is one of the most important concepts in most scientific fields. Many cultures regard time and space as the fundamental scale of the world. The discussion about time has started along with the culture of human being. In the field of the Semantic Web, time is not an exception. Semantic Languages more or less support the temporality concepts. In this section, we first discuss some basic ideas of time ontology by following Ermolayev et al 2014 paper [19], and then we discuss further about how W3C defines time ontology in their OWL2 standards [20].

3.1 Time Ontology Review

Ermolayev et al 2014 paper [19] summarized a taxonomy of temporal features as shown in the graph below. Although the authors state that this taxonomy might be incomplete and some fragments are also questionable, we still think this taxonomy is the best one
we can follow by far. Under the root category as TemporalEntity, four sub categories are discussed: TemporalRule, TemporalConcepts, TemporalFeature, and TemporalProper-Measure. The completeness of this review may beyond the scope of temporality in semantic web, but it is benefit for us to get a complete understanding of the time in general at first, then to get a deep understanding of the time in semantic web in specific later on.

3.1.1 Temporal Features

The taxonomy of Temporal Features is shown as Fig.4 below. There are several important facets of time require sufficient discussion.

1. Unbounded versus Bounded Time. This is actually more like a philosophical discussion; and in summary as Koubarakis and Iwasaki et al did, we take the view that time is unbounded both in the past and in the future and not cyclic [24, 25, 26, 27]. In addition, based on convenience, in some application, a certain time point can be chosen to be time zero as the start of the time. This doesn’t conflict unbounded past time but indicates the time before the time zero doesn’t relevant to the application.
2. Anisotropy of Time. In summary, time is regarded anisotropic that only present is real, time has a direction, and time flows in that direction [24, 28, 29, 30].

3. Partitioning and Structuring of Time. A time line is here to be very useful to cover both the point view and the interval view of time [25, 27, 31]. In the mixed point-interval-based framework, based on their position on the time line, several structural elements are pictured as following.

4. Density of Time. Depends on the specific programs, the model of time might be discrete which time is sparse, or be real number which time is continuous, or dense number sets [22, 26, 29, 32, 33, 34].

5. Order of time: Linear versus Branching. Two aspects related to a theory of time lead to the debate of linear versus branching time: one is temporal incidence and the
other is the order of time. While most of the theories standing for the branching time, many permit the co-existence of both by allowing many parallel time lines.

6. Temporal Uncertainty. This aspect indicates that time of occurrence is uncertain; and this uncertainty can be reduced by constraints. However, although the preciseness can be improved, the exact time is still blur [35].

7. Periodicity in Time. Many event occur periodically in nature. The example is the clock which repeats in a circular manner. Most theories ignore this concept while Ermolayev et al 2008 paper uses sets of periods to represent periodic time structure. A period is defined as time intervals corresponding to the occurrences of regular or repeating events.

3.1.2 Temporal Elements and Structures

For describing time, we need to use Time Elements and Time Structure, which is represented in the taxonomy in Fig.6 below.

![Temporal elements and structures](image)

A temporal element is the primitive entity that is used to represent time. These elements for a theory of time are Time Points and Time Intervals as Fig.7 shows. A temporal structure is defined as a compound construct in a theory and is built of temporal structures and elements for a particular purpose, such as a calendar.
Time Points are the elements of a time line which can also be called instants. Time point (instant) $t$ has no duration. The value of $t$ differs in the specific programs. Some assign non negative values to an instant, some assign negative value to represent past and positive to represent future.

Time Intervals are used to extend the point-based model of time to represent a period or a part of the time. Also, the relationship "during" is associated with a duration, which cannot be time points, but be an interval. Some theories argue that time interval is a compound structure of time points.

3.1.3 Temporal Properties

Fig.8 represents the taxonomy of the temporal properties as below.

Dates and Timestamps both are widely used in programs. Dates are ordered triples of integers for year, month, and day. Timestamps could be interpreted as a metric reference to a point in the time line. Timestamps can contain precise information and the preciseness is depended on the system.
3.1.4 Temporal Relations

Temporal Relations are the binary properties among elements of a time model. Depends on the elements in the relationship, there are three categories: point to point, point to interval, and interval to interval. Fig.9 represents the taxonomy of the temporal relations as below.

Fig. 8. Temporal properties

Fig. 9. Temporal relations
3.2  Time Ontology Development

In this section, we follow Hobbs and Pan's paper "An ontology of time for the semantic web" [21] to show how to build an simple ontology of time. To prevent simply repeating every details, we only include key definitions and explanations here.

3.2.1  Temporal Relations

1. TemporalEntity class has two and only two subclasses: Instant(t), and Interval(T).

2. A Interval(T) has two instant(t) t1, and t2 as begin point and end point. Normally, to be a proper interval, t1 will be different than t2. And proper interval is assumed for following discussion about intervals.

3. inside is a relationship between an instant and interval.

4. Before relation on temporal entities T1 and T2, we compare the end of T1 with the start of T2. For above relationship, if an instant t is inside an Interval T, then the start of T should be before t, and t should be before the end of T.

5. After is the reverse relationship of Before, If T1 is before T2, then T2 is after T1.

6. Interval relations can be defined using Before and the start and end points of Intervals. Say, Interval T1 starts at t1 and ends at t2, and Interval T2 starts at t3 and ends at t4. Then we can have the relationship between T1 and T2 as defined below:
   T1 equals T2 if t1 equals t3 and t2 equals t4;
   T1 meets T2 if t2 equals t3;
   T1 overlaps T2 if t3 before t2, and t2 before t4;
   T1 starts T2 if t1 equals t3;
   T1 During T2 if t1 before t3 and t4 before t2;
   T1 finishes T2 if t2 equals t4;

This relationship is summarized in a graph by W3C as shown in Fig.4 of next section.
3.2.2 Linking Time and Events

There are four predicates that make the linkage between time ontology and events. Let e be an event, then:

- atTime(e, t) indicates e happens at instant t;
- during(e, T) indicates e happens during interval T;
- holds(e, t) indicates atTime(e, t) if t is an instant and during(e, T) if t is an interval;

3.2.3 Measuring Durations

To measure durations the most convenient way is mapping intervals as units of time to reals such as:

- minutes([5:14, 5:17]) = 3;
- years([2014, 2016]) = 2.

and the arithmetic relations among units of time follow the nature facts such as:

- minutes(T) = 60*seconds(T);
- hours(T) = 60*minutes(T).

3.2.4 Clock and Calendar

This subsection we are going to discuss widely accepted facts of temporal elements: Time Zones, Clock and Calendar Units, Weeks, Months and Years, etc.

1. Hobbs and pan have developed a time zone resource in OWL that links a simple geographic ontology with the time ontology. It handles all the usual time zone and daylight saving, as well as unusual cases.

2. Concepts as duration may differ than concepts (although they share same name) in calendar. For example, day in calendar is 0:00am to 23:59pm; however, day in duration can be any 24 hour length interval. Furthermore, in calendar interval, we always include the beginning point and exclude the ending point.

3. A week is any seven-day length interval; but a calendar week, is starting from Sunday (or Monday). Weeks are independent of month and year, but we can define a week in a particular month and year to be nth week of the month and year.

4. Months may have different names in description. For example, Month 2 is called February in English.
3.2.5 Describing Times and Durations

There are various ways of describing times and durations. Here we discuss the most commonly accepted ones, Timestamps, Calendar-Clock Descriptions, and Duration Descriptions.

1. Timestamps is a standard notation for describing time as year, month, day, hour, min, second, and timezone. For instance, 2:29:22am PST, Thursday, Dec22, 2016 can be recorded in timestamp format as timeOf(t, 2016, 12, 22, 2, 29, 22, PST).

2. Calendar-clock description methods uses a set of properties: unitType, yearOf, monthOf, etc to express a time. For instance, the above time will be expressed as unitType(seconds), yearOf(2016), monthOf(12), dayOf(22), hourOf(2), minutesOf(29), secondsOf(22), timeZoneOf(PST).

3. Duration Description may differ if using different units. For instance, 1 day 2 hours will be equivalent with 26 hours, or 1560 minutes. More details can be found in Hobbs and Pan's paper [21] for reference.

3.3 Time Ontology in OWL

The basic structure of the time ontology and temporal relations are based on Allen and Ferguson's binary relations on intervals [22, 23]. The top class TemporalEntity has two subclasses Interval and Instant (as the point we discussed above) [20]. The core model of temporal entities is shown as below:

OWL2 has two built-in datatypes relating to time, which are date and timestamp. This is also consistent with the previous review for description time.

The relations between time intervals are adopted from Allen's paper too. They are before, meets, overlaps, etc as shown in the graph below:
4 Temporal Database

As we discussed, the Semantic Web, especially the Linked Data Initiative, is to make the WWW as a big database. Since the field of Temporal Database has been explored a lot in the past, it will be beneficial we borrow from that field, and make a connection between Temporality in database and Temporality in Semantic Web.

Time is a very important aspect in our real world. It is the most commonly accepted dimension. Almost everything can be attached with a timestamp to make a snapshot of its current state so that we can not only record what it is "now", but also what it was in the "past", and what it will be in the "future".
A good database should record the real world in a reliable and efficient way. The demand for including temporal data in the database is increasing, such as airline reservation, hotel and conference room reservation, inventory system, medical system, accounting, etc. Temporal database system is important today, and will be more and more important in the future.

4.1 Concept of Temporal Database

Classical database system, is good at managing data without the dimension of time. Everything is recorded as a current true statement. If it comes to be false, it will be deleted from the database. If it comes to be true again, it will be inserted into the database again. However, this time-free style cannot satisfy the need of users. For example, we cannot reserve a room without an attribute of time because, a room is only reserved for a time period. The statement becomes true when it is in the period, and false if it is not. We cannot just add a record that the room is reserved at the beginning of the time period and remove this record at the end of the period. We need time as attributes to reflect the fact that although it is not true now (before the period), but it will be true in the period, and it will be false again after that. So, we can use this room at other time that is not conflict with the reserved period. Thus, besides the attributes we used in the classical database system, time should be included. In addition, in classic database system, if an information is updated, a new record will replace the old one and we cannot keep both records, thus, the change cannot be observed.

In terms of formal definition of temporal database, we need to define time associated data first:

A Valid Time is a time period during which the statement / record is true with respect to the real world. For example, William Weber is a graduate student of CUNY in 2009-2015. Then the valid time of statement "William Weber is a graduate student of CUNY" should be (2009-2015). This time period can be satisfied in a database by using temporal attributes such as "beginOfProgram" and "endOfProgram", or "attendProgram" and "graduateDate", or "inProgram" interval, etc. The temporal attributes are independent of the transaction time in the database; they are reflecting the time in real world.

A Transaction Time is a time period during which the statement / record in the database is considered to be true. It is associated with the time that the statement is added to the database and is assume to be true till now. For the same example of William Weber, a
statement should be inserted into the database in 2009 as William Weber is in the program; and need to be delete in 2015 to reflect the fact that William Weber is graduated and that statement is no longer true. The transaction time (2009 and 2015) is the attribute in the database system, and representing the statement is true to present; or it needs to be removed to reflect the change.

For example, we have a story for a company that has several employees: William, Tom, Jack, Rose, Mary. The EmployeeID, Name, Department, and Salary are shown as table below in relationship DBMS:

Table 1: Employee Information Version 1

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>Depart</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>120,000</td>
</tr>
<tr>
<td>13</td>
<td>Tom</td>
<td>Admin</td>
<td>115,000</td>
</tr>
<tr>
<td>33</td>
<td>Jack</td>
<td>Sale</td>
<td>80,000</td>
</tr>
<tr>
<td>43</td>
<td>Mary</td>
<td>Accounting</td>
<td>65,000</td>
</tr>
</tbody>
</table>

Employee information represented in Relational DBMS

From the above table, what we can conclude is that these five people are in which department with how much salary. We don’t know what salary they had before, and which department they belonged before. We also don’t know Tim (who is not on the table) has worked for the company before or not. This table only shows current state of company’s employee information.

A temporal database is the database that has temporal data, and is able to deal with insertion, deletion, and query for temporal data. In terms of the two major types of time - valid time and transaction time, temporal database has three forms: historical that supports only valid time, rollback that supports only transaction time, and bi-temporal that supports both [51].

4.1.1 Historical Database

Historical database only supports valid time. Thus, it can be regarded as classical relational database with a new temporal attribute represents the valid time. In a historical Database, above Employee Information may be described as below:
Table 2: Employee Information Version 2

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>Depart</th>
<th>DepFrom</th>
<th>DepEnd</th>
<th>Salary</th>
<th>SalFrom</th>
<th>SalEnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>2004</td>
<td>NOW</td>
<td>60,000</td>
<td>2004</td>
<td>2005</td>
</tr>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>2004</td>
<td>NOW</td>
<td>75,000</td>
<td>2006</td>
<td>2008</td>
</tr>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>2004</td>
<td>NOW</td>
<td>100,000</td>
<td>2009</td>
<td>NOW</td>
</tr>
<tr>
<td>13</td>
<td>Tom</td>
<td>Admin</td>
<td>2007</td>
<td>NOW</td>
<td>80,000</td>
<td>2007</td>
<td>2010</td>
</tr>
<tr>
<td>13</td>
<td>Tom</td>
<td>Admin</td>
<td>2007</td>
<td>NOW</td>
<td>100,000</td>
<td>2011</td>
<td>NOW</td>
</tr>
<tr>
<td>33</td>
<td>Jack</td>
<td>Sales</td>
<td>2004</td>
<td>NOW</td>
<td>100,000</td>
<td>2009</td>
<td>2010</td>
</tr>
<tr>
<td>33</td>
<td>Jack</td>
<td>Sales</td>
<td>2004</td>
<td>NOW</td>
<td>130,000</td>
<td>2011</td>
<td>NOW</td>
</tr>
<tr>
<td>41</td>
<td>Rose</td>
<td>Sales</td>
<td>2009</td>
<td>2011</td>
<td>110,000</td>
<td>2009</td>
<td>2011</td>
</tr>
<tr>
<td>43</td>
<td>Mary</td>
<td>Accounting</td>
<td>2012</td>
<td>NOW</td>
<td>100,000</td>
<td>2012</td>
<td>NOW</td>
</tr>
</tbody>
</table>

Employee information represented in Historical Database

Now we can conclude with much more information. For example, we know that William stays in Admin department since 2004, and his salary has been increased twice: his salary was 60,000 in 2004 to 2005, 75,000 in 2006 to 2008, and 100,000 from 2009 to now. In addition, Tom was in HR department before he joined Admin, and his salary changed twice as well. Furthermore, Rose was in sales department from 2009 to 2012, and she is not in any department now. We may conclude that she has left the company. As we can see, historical database adds 4 extra columns to represent the valid time attributes, which makes the database more complex, however, it provides more power for recording information.

4.1.2 Rollback Database

Rollback database is the another type of database. It is different than the historical database and only supports transaction time. Thus, it keeps all records with their timestamps when they are entered. Rollback database is ideally for data recovery from a failure [51]. In a rollback database, the Employee Information may be described as below:

From above table, we can interpret similar information as Historical Database has. For example, we added William with salary 60,000 in 2004; and updated it with 75,000 in 2006. Then we can conclude William's salary is 60,000 from 2004 to 2005. How-
Table 3: Employee Information Version 2

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>Depart</th>
<th>Salary</th>
<th>RecordDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>60,000</td>
<td>2004</td>
</tr>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>75,000</td>
<td>2006</td>
</tr>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>85,000</td>
<td>2008</td>
</tr>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>75,000</td>
<td>2008</td>
</tr>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>100,000</td>
<td>2009</td>
</tr>
<tr>
<td>13</td>
<td>Tom</td>
<td>HumanResource</td>
<td>60,000</td>
<td>2004</td>
</tr>
<tr>
<td>13</td>
<td>Tom</td>
<td>Admin</td>
<td>80,000</td>
<td>2007</td>
</tr>
<tr>
<td>13</td>
<td>Tom</td>
<td>Admin</td>
<td>100,000</td>
<td>2011</td>
</tr>
<tr>
<td>33</td>
<td>Jack</td>
<td>Sales</td>
<td>100,000</td>
<td>2009</td>
</tr>
<tr>
<td>33</td>
<td>Jack</td>
<td>Sales</td>
<td>130,000</td>
<td>2011</td>
</tr>
<tr>
<td>41</td>
<td>Rose</td>
<td>Sales</td>
<td>110,000</td>
<td>2009</td>
</tr>
<tr>
<td>43</td>
<td>Mary</td>
<td>Accounting</td>
<td>100,000</td>
<td>2012</td>
</tr>
</tbody>
</table>

Employee information represented in Rollback Database

However, we changed the salary to 85,000 and then changed it back to 75,000 in 2008 which isn't in historical database. There maybe two possibilities: the first one is the historical database failed to reflect the change inside a year. Because when a new record is added into the database with same valid time, the old one will be deleted. The second one is the database manager added a record wrongly and soon changed it back.

We can also find out from above table, that it is very easy to retrieve data back if we need to. We can rollback to a time point and ask "what salary William has in 2008 as if we were in 2008".

4.1.3 Bi-Temporal Database

Bi-temporal database is the one that most used temporal database in reality. This database supports both types of time. Bitemporal data combines both Valid and Transaction time. For example, the statement "William Weber is a graduate student of CUNY" will have two temporal attributes. The first is the valid time that the statement is true in real word 2009 to present. The second one is the transaction time that the statement is considered to be true when it is added into the database in 2009. And in stead of deleting this record in 2015 when William is graduated, we have another record to reflect the
change that is "William Weber is graduated" that has valid time as 2015 to present and transaction time as 2015. The table below represents what the Employee Information data will be in Bi-temporal database.

Table 4: Employee Information Version 4

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>Depart</th>
<th>DepF</th>
<th>DepE</th>
<th>Salary</th>
<th>SalF</th>
<th>SalE</th>
<th>RTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>2004</td>
<td>NOW</td>
<td>60,000</td>
<td>2004</td>
<td>NOW</td>
<td>2004</td>
</tr>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>2004</td>
<td>NOW</td>
<td>60,000</td>
<td>2004</td>
<td>2005</td>
<td>2006</td>
</tr>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>2004</td>
<td>NOW</td>
<td>75,000</td>
<td>2006</td>
<td>NOW</td>
<td>2006</td>
</tr>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>2004</td>
<td>NOW</td>
<td>75,000</td>
<td>2006</td>
<td>2007</td>
<td>2008</td>
</tr>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>2004</td>
<td>NOW</td>
<td>85,000</td>
<td>2007</td>
<td>NOW</td>
<td>2008</td>
</tr>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>2004</td>
<td>NOW</td>
<td>75,000</td>
<td>2006</td>
<td>NOW</td>
<td>2008</td>
</tr>
<tr>
<td>12</td>
<td>William</td>
<td>Admin</td>
<td>2004</td>
<td>NOW</td>
<td>75,000</td>
<td>2006</td>
<td>2008</td>
<td>2009</td>
</tr>
<tr>
<td>13</td>
<td>Tom</td>
<td>HR</td>
<td>2004</td>
<td>NOW</td>
<td>60,000</td>
<td>2004</td>
<td>NOW</td>
<td>2004</td>
</tr>
<tr>
<td>13</td>
<td>Tom</td>
<td>HR</td>
<td>2004</td>
<td>2006</td>
<td>60,000</td>
<td>2004</td>
<td>2006</td>
<td>2007</td>
</tr>
<tr>
<td>13</td>
<td>Tom</td>
<td>Admin</td>
<td>2007</td>
<td>NOW</td>
<td>80,000</td>
<td>2007</td>
<td>NOW</td>
<td>2007</td>
</tr>
<tr>
<td>13</td>
<td>Tom</td>
<td>Admin</td>
<td>2007</td>
<td>NOW</td>
<td>80,000</td>
<td>2007</td>
<td>2010</td>
<td>2010</td>
</tr>
<tr>
<td>13</td>
<td>Tom</td>
<td>Admin</td>
<td>2007</td>
<td>NOW</td>
<td>100,000</td>
<td>2011</td>
<td>NOW</td>
<td>2011</td>
</tr>
<tr>
<td>33</td>
<td>Jack</td>
<td>Sales</td>
<td>2004</td>
<td>NOW</td>
<td>100,000</td>
<td>2009</td>
<td>NOW</td>
<td>2009</td>
</tr>
<tr>
<td>33</td>
<td>Jack</td>
<td>Sales</td>
<td>2004</td>
<td>NOW</td>
<td>100,000</td>
<td>2009</td>
<td>2010</td>
<td>2011</td>
</tr>
<tr>
<td>41</td>
<td>Rose</td>
<td>Sales</td>
<td>2009</td>
<td>NOW</td>
<td>110,000</td>
<td>2009</td>
<td>NOW</td>
<td>2009</td>
</tr>
<tr>
<td>41</td>
<td>Rose</td>
<td>Sales</td>
<td>2009</td>
<td>2011</td>
<td>110,000</td>
<td>2009</td>
<td>2011</td>
<td>2012</td>
</tr>
<tr>
<td>43</td>
<td>Mary</td>
<td>Accounting</td>
<td>2012</td>
<td>NOW</td>
<td>100,000</td>
<td>2012</td>
<td>NOW</td>
<td>2012</td>
</tr>
</tbody>
</table>

Employee information represented in Bi-temporal Database

As we can see from above table, the bi-temporal database represents the most complete information. We can have the valid time for each statement, also keep the history of the statement using transaction time.

4.1.4 Connection Between Temporal Database and Relational DBMS

There are multiple proposals for building a temporal database from a relational database. Some are extending the structure of the relational database for temporal data. In these
approaches, we can leave the relational databases as what they are without any change. However, the database application programmer has to take the responsibility to handle temporal data. For example, we can create a data type for time, date, and regard this new type as any other types of data. Even further, we can create a new abstract database type (ADT) as "time".

Some others make changes to existing relational databases. For example, we can get time-stamping data by extending non-temporal schemas or types with time as attributes. This approach only changes a part of the relational database and is the most widely adopted. Going further, some try to generalize the whole data model to support temporal data. This approach will change all of the relationship database, not only data structures, but also operations and integrity constraints, will be generalized [52].

4.1.5 Time-stamping Approach

There are two approaches for time-stamping in DBMS. One is tuple time stamping, the other is attribute time-stamping. Tuple time stamping approach keeps the 1NF relations, and adds timestamps to each tuple in a relation. The tables we discussed above for the employee information are using tuple time stamping approach. As we can see, each time we update an attribute associated with time, we have to insert a new tuple into the table, which will produce a lot of redundant information. If there are many time-related attributes, the redundancy will be significant. The advantage of tuple time stamping is minimum change needed for existing relation database. Since we can consider a time related attribute as a regular one, and we keep the 1NF relation, we can use the structure and operations directly. Because of the simplicity of implementation, application developers and users can understand the structures of a temporal database and conduct queries as they do in a commercial relational database.

Attribute time-stamping uses a more complex way to store the temporal elements. When an attribute has a temporal elements, it will be attached onto, no matter the elements are time points, intervals, etc. Instead of a simple attribute, it becomes a relation that expresses the attribute value with associated temporal elements. Thus, attribute time-stamping also called nested time stamping, because we can consider the temporal attribute as a sub-relation of the tuple relations. We show the employee information table with attribute time stamping as below:
### Table 5: Employee Information Version 5 (Partial for William only)

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>Depart</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>⟨[2006, 2008), [2006, 2008), 75,000⟩</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>⟨[2007, 2008), [2008, 2009), 85,000⟩</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>⟨[2007, 2008), [2008, 2009), 75,000⟩</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>⟨[2009, NOW), [2009, NOW), 100,000⟩</td>
</tr>
</tbody>
</table>

Bi-temporal Database using Attribute Time-Stamping.

### 4.2 Temporal Relational Algebra

#### 4.2.1 Preliminaries

**Time**  
Time is a continues variable; however, DBMS record time as discrete points for practical reasons. If the origin of time is \( t_0 \) and current time is \( t_{\text{now}} \), time \( T \) can be represented as a set of \( \{ t_0, t_1, t_2, ..., t_{\text{now}} \} \), and \( t_0 < t_1 < ... < t_{\text{now}} \), and \( t_0 + 1 = t_1, ..., t_{\text{now}-1} + 1 = t_{\text{now}} \).

**Point and Interval**  
Time points and intervals both represent time variable; and can be converted into each other. Time points can be regarded as the smallest interval; and time interval can be regarded as a set of time points. We use \([l, u)\) to represent the interval that starts at \( t_l \) (lower bound) inclusive and ends at \( t_u \) (upper bound) exclusive. Thus, \( t_4 \) can be represented as interval as \([4, 5)\), and \([4, 7)\) can e represented as a set of time points \([t_4, t_5, t_6)\). Since they are interchangeable and interval representation can save more space, we just use interval representation from now on.

**Attribute**  
An attribute may have temporal element. If the attribute \( A \) is time invariant, such as Social Security Number, the value itself forms an atom. If \( A \) is time variant, that means the value \( a \) of \( A \) is valid for a certain time and not valid for other time, the attribute needs to include the temporal element as \( ⟨\langle l, u⟩, a⟩\).

#### 4.2.2 Historical Relational Algebra

This subsection will be based on [53].
Model Let $U$ be the set of all possible attribute values such as integers, real numbers, strings, and null. Let $D_{a1}, ..., D_{am}$ be subsets of $U$ and $D_{s1}, ..., D_{sm}$ be subsets of $P(U)$ where $P(U)$ is the powerset of $U$. $T_i$ is the set of time points in $T$ and it is the subset of $T \times T$ where $\times$ denotes the Cartesian product: $T_i = \{(l, u) | l < u \land t_0 \leq l < now \land t_0 \leq u < now \land l, u \in T \times T\}$. $D_{t1}, ..., D_{tm}$ are the subsets of $T_1 \times U$ and $P(D_{t1}), ..., P(D_{tm})$ are their corresponding power sets. A collection of sets $E_1, ..., E_n$ where $E_i$ is one of the sets $D_{a1}, ..., D_{am}, D_{s1}, ..., D_{sm}, D_{t1}, ..., D_{tm}, P(D_{t1}), ..., P(D_{tm})$ for $i = 1, ..., n$. A historical relation is a subset of the Cartesian product $E_1 \times E_2 \times ... \times E_n$. Thus, the attribute values can only be sets, not sets of sets and so on. We call $R(A_1, ..., A_n)$ a historical relation scheme and $n$ is called the degree of $R$. $At\!r(R)$ is the set of the attributes of $R$. $r$ denotes the instance of $R$.

Notations The discussion in this section shares the following notation. Let $R$ and $S$ be two relations, and $X$ and $Y$ be lists of attributes, and $|X| = |Y| = n \geq 1$, $X \subseteq At\!r(R)$ and $Y \subseteq At\!r(S)$. $X \theta Y$ denotes the every attribute in the list operate $\theta$ as $X_1 \theta Y_1 \land X_2 \theta Y_2 \land \ldots \land X_n \theta Y_n$. Type($X_i$) denotes the type of the $i^{th}$ attribute of $X$ and the underlying domain. Type($X_i$) = Type($Y_i$) indicates the types of the $i^{th}$ attribute of $X$ and $Y$ and their underlying domain are the same. Thus, $X_i$ and $Y_i$ are union compatible. Type($X$) = Type($Y$) indicates that every attribute of $X$ is the same as the attributes of $Y$. For the four attribute types of $A$, $A_d$ denotes the set of atomic attributes in $R$. $A_s$ denotes the set of set-valued attributes in $R$. $A_t$ denote the sets of triplet-valued and $A_{st}$ denotes set-triplet-valued attributes. For the operators $\theta$, $\theta_r$, denotes the set of relational comparison operators $\{=, \neq, >, \geq, <, \leq\}$. $\theta_s$ denotes the set of set comparison operators $\{\in\}$. And $\theta_m$ is the set membership operator $\{\in\}$.

Attribute Types There are four types of attributes in $R$. Atomic attributes contain atomic values which are subsets of $U$. Triplet-valued attributes contain triplets as the format of $\langle l, u, a \rangle$. Set-valued attributes contain a set of atomic values and set-triplet-valued attributes contain sets of triplets as values. Each set contains one or more triplets and represents the history of the attribute over $T$.

Standard relational algebra operations Standard relational algebra operations can be applied to Historical Relations directly with minor modifications. Projection ($\pi$) and Cartesian product ($\times$) operations are remain unchanged. Set union ($\cup$) and Set difference ($-$) may have overlapping, adjacent, and contained intervals and then change the attribute type from triplet-valued to set-triplet-valued. In order to handle different attribute types selection ($\sigma$) should be modified as below:
(a) $\theta \in \theta_r$, when $X, Y \in A_o$ or $X \in A_t, Y \in A_t$. When triplet-valued attributes are referenced, their components should be used such as $Y_l, Y_u, X_l, X_u, Y_v$ or $X_v$. For example, $X_u < Y_l \land X_v = Y_v$ indicates that interval part of $X$ is before the interval part of $Y$ and their value part is equal.

(b) $\theta \in \theta_s$, when $X, Y \in A_s$ or $X, Y \in A_{st}$.

(c) $\theta \in \theta_m$, when $(X \in A_o \lor X \in A_t) \lor Y \in A_s, or X \in A_t and Y \in A_{st}$. If $X$ is a triplet-valued attribute, its components $X_l, X_u, X_v$ should be used.

Join ($\triangledown \bowtie$) can be defined similarly as selection above.

**Aggregate formation operation**  Let $R$ be a relation and $X \subseteq \text{Attr}(R)$ with $|X| = k$. For the atomic or triplet-valued attribute $A$ in $R$ and aggregate function $f$, aggregate formation operation is defined as:

$$R(X, f_A) = \{t[X] \ y | t \in T \land y = \sum_{t' \in R \land t[X] = t'[X]} f_A(t')\}$$

The result is a relation of degree $k + 1$.

**New and revised operations**  Triplet-valued attributes can be packed to a set-triplet-valued attributes and unpack does the reverse. Atomic attributes can form triplet-valued attributes by triplet-formation to convert three attributes together while triplet-decomposition break triplet-valued attributes to its components. Slice operation restricts the time of an attribute based on the time of another attribute. Drop-time operation discards the time component and only keep the atom or atom set from triplet-valued attributes or set-triplet-valued attributes. These operations are presented in the below graph:
Pack ($P$) and unpack operation ($U$) The pack operation ($P$) collects the values in attribute $A$ into a single tuple component whose remaining values agree. If Relation $R$ has degree of $n$ and its attribute $A$, for each $(n-1)$ tuple in $\pi_C(R)$, an $n$-tuple $W_g$ is defined as:

$$W_g[C_A] = g$$

$$W_g[A] = \begin{cases} [t[A]] & \text{if } A \in \{A_l, A_r\} \\ \{x | \exists t \in R : t[C_A] = g \land x \in t[A]\} & \text{if } A \in \{A_s, A_{st}\} \end{cases}$$

Then $P_A(R) = \{W_g | g \in R[C_A]\}$

The unpack operation ($U$) creates a family of tuples for each tuple of the relation $R$'s set-valued attributes. Let $t$ be a tuple of $R$, then:

$$U_A[t] = \begin{cases} \{t\} & \text{if } A \in \{A_l, A_r\} \\ \{t' | t'[A] \in t[A] \land t'[C_A] = t[C_A]\} & \text{if } A \in \{A_s, A_{st}\} \end{cases}$$

Then $U_A(R) = \bigcup_{t \in R} U_A(\{t\})$

Triplet-decomposition (T-DEC) and triplet-formation operation (T-FORM) Triplet-decomposition breaks a triplet-valued attribute $\bar{A}$ into its components. It adds two new attributes for the lower and upper bound interval and replace the triplet-valued attribute $\bar{A}$ with its original value component $\bar{A}_v$. Let degree($R$) = $n$ and $A$ is one of $R$'s attributes. Triplet-decomposition (T-DEC) creates a new relation $R'$ whose degree is $n + 2$.

$$\text{T-DEC}_A(R) = \{t, l, u | (\exists t')(t' \in R \land t[C_A] = t'[C_A] \land t[A] = t'[\bar{A}_v] \land t' = t'[,L] \land u = t'[\bar{A}_u]\}$$

The new attributes are $A_L$ and $A_U$ that are the $(n + 1)^{st}$ and $(n + 2)^{nd}$ attribute of $R'$ and corresponding to the $\bar{A}_l$ and $\bar{A}_u$ in $R$.

Triplet-formation operation (T-FORM) creates a triplet-valued attributes from three attributes $A$, $L$, and $U$ which compose the value, lower and upper bound of the interval for a triple-valued attribute $A$. Let $R$ be a relation and degree($R$) = $n + 2$, $A, L, U \in Atr(R)$, and $C_A = Atr(R) - \{A, L, U\}$, then:

$$\text{T-FORM}_{A,L,U}(R) = \{t | (\exists t')(t' \in R \land t[C_A] = t'[C_A] \land t[A_v] = t'[A] \land t[A_l] = t'[L] \land t[A_u] = t'[U]\}$$

The resulting relation has degree of $n$. 

49
Slice operation (SLICE)  Slice operation slices the triplets of an attribute $\bar{A}$ based on the time of another attribute $\bar{B}$. This operation checks the interval portions of the time component of two attributes to determine whether or not they overlap. If they overlap, the intersection of time component of the two attributes will be the time component of the result, and value component of $\bar{A}$ will be the value component of the result. If they do not overlap, no triplet will result from this operation. Let $\bar{A}$ and $\bar{B}$ be two triplet-valued attributes of relation $R$:

$$\text{SLICE}_{\bar{A}, \bar{B}}(R) = \{ t | (\exists t^\prime)(t^\prime \in R \land t[C_{\bar{A}}] = t^\prime[C_{\bar{A}}] \land t[\bar{A}_v] = t^\prime[\bar{A}_v] \land t[\bar{A}_l] \leq t^\prime[\bar{B}_l] \land t[\bar{A}_u] \leq t^\prime[\bar{B}_u]) \}$$

Drop-time operation (DROP-TIME)  Drop time operation discards the time components of a triplet-valued or a set-triplet-valued attributes. These attributes will be converted into an atomic or a set-valued attribute. The difference between triplet-decomposition operation and drop time operation is, former keeps the time component while later discards them. Let $R$ be a relation and $A \in \text{Attr}(R)$, then:

$$\text{DROP-TIME}_A[R] = \begin{cases} 
\{ t | (\exists t^\prime)(t^\prime \in R \land t[C_{\bar{A}}] = t^\prime[C_{\bar{A}}] \land t[A] = t^\prime[\bar{A}_v] \} & \text{if } A = A_t \\
\{ t | (\exists t^\prime)(t^\prime \in R \land t[C_{\bar{A}}] = t^\prime[C_{\bar{A}}] \land x \in t^\prime[*\bar{A}] \land x_v \in t[A] \} & \text{if } A = A_{st} \\
R & \text{otherwise}
\end{cases}$$

4.2.3 Nested Temporal Algebra

This subsection is intended to discuss the algebra used for temporal relations. [49] will be used as a case study here. We category the temporal relational algebra operations to basic operations and derived operations; and in this subsection, we use the basic operations to illustrate the idea of temporal relational algebra.

There are eight basic operations in: Set Operations, Projection, Selection, Cartesian Product, Unnest, Nest, Temporal Atom Decomposition, and Temporal Atom Formation. While Set Operation, Projection, Cartesian Product are exactly the same as relational algebra, others need modifications. The nested temporal model is a generalized model of the historical relational model where the latter one has order 1. Thus, the algebra operations remain the same except the nesting and unnesting operations. These operations will be discussed in next section.
4.2.4 Nested Bitemporal Relational Algebra

This subsection will be based on [54].

Bitemporal Atom  As we discussed previously, a temporal atom is a \( \langle t, a \rangle \) pair, where \( t \) is a temporal interval \([l, u]\) that represents the start and end time that value \( a \) is valid. A bitemporal atom has two temporal intervals to represent not only the valid time of the data value but also the transaction time of it. Thus, \( t \) in a bitemporal relation has a format as \( \langle TT, VT, V \rangle \). Also, since time points and time intervals can be converted from one to the other easily; and time interval is more general, we still use time interval to represent the transaction time and valid time in a bitemporal atom as \( \langle [TT_l, TT_u], [VT_l, VT_u], V \rangle \).

Nested Bitemporal Relation  The nested relation schema is defined reductively on the nesting depth, or order, of a schema. An order zero schema is an atom or a temporal atom. A tuple schema is composed by a sequence of attributes that are atoms, temporal atoms, or nested relations. The order of a tuple schema is the same as the maximal order of its components. A bitemporal relation schema is defined over a tuple schema and has one more order than the tuple. The inductive definition of bitemporal tuple and bitemporal nested temporal relation schemas is:

Order Zero Schema: \( t := \text{tuple} :< t(1), ..., t(n) > \) for \( n > 0 \). Each \( t(i) \) is an atom or a bitemporal atom.

Order \( K + 1 \) Schema: Either a bitemporal tuple schema or a bitemporal nested relation schema.

i. \( t := \text{tuple} :< t(1), ..., t(n) > \) for \( n > 0 \). Each \( t(i) \) is an atom, a bitemporal atom, or a bitemporal nested relation schema of order \( k + 1 \) or less. At least one component of \( t \) must be a relation schema of order \( k + 1 \).

ii. \( t := \text{relation} :< t > \) where \( t \) is the bitemporal tuple schema \( < t(1), ..., t(n) > \). Each \( t(i) \) has a schema of order \( k \) or less and at least one \( t(i) \) has a schema of order \( k \). For simplicity, we just write \( r := \text{relation} :< t(1), ..., t(n) > \).

Context  A bitemporal database has both valid time and transaction time elements, thus, it has three contexts. Current context only covers currently valid tuples of a bitemporal relation. Historical context allows rollback a bitemporal relation to a given time. Notation \( R_{t,X} \) refers the state of \( X \) attribute in a bitemporal relation \( R \) at a given time \( t \). Bitemporal context covers both current and historical context.
For following discussion upon bitemporal context, we use \( \text{RelationE} : < e(1), ..., e(n) > \) to represent a bitemporal relational algebra expression \( E \). \( \text{EV}(E) \) represents the evaluation of \( E \).

**Set Operations** (\( \cup, \cap, - \))  
Set operations of a bitemporal relation requires no change on relational operations.

\[
\text{EV}(R \cup S) = \{ t \mid t \in \text{EV}(R) \lor t \in \text{EV}(S) \}.
\]

\[
\text{EV}(R \cap S) = \{ t \mid t \in \text{EV}(R) \land t \in \text{EV}(S) \}.
\]

\[
\text{EV}(R - S) = \{ t \mid t \in \text{EV}(R) - t \in \text{EV}(S) \}.
\]

**Projection** (\( \pi \))  
The projection operation will eliminates duplicate tuples and is defined as below:

If \( X \subseteq \text{atr}(E) \) then \( \text{EV}(\pi_x(E)) = \{ s[x] \mid s \in \text{EV}(E) \} \).

**Selection** (\( \sigma \))  
We define a formula \( F \) as a format of \( iop j \) where \( op \) is one of \{\( =, \neq, <, \leq, >, \geq \)\} and \( i \) and \( j \) are attribute names or index showing the position of attributes in relation \( E \). Then:

\[
\text{EV}(\sigma_F(E)) = \{ s[x] \mid s \in \text{EV}(E) \land F \text{ is true} \}.
\]

Note that if attribute \( A \) has values of bitemporal atoms, then \( A.TT, A.VT, \) and \( A.V \) refer to the transaction time, valid time, and value components of \( A \). Additionally, \( A.TT_l, A.TT_u, A.VT_l, A.VT_u \) refer to the lower and upper bound of the transaction and valid time interval. For operation \( F \), while equality (\( = \)) and inequality (\( \neq \)) can be applied to all atom types (values atoms, bitemporal atoms, sets), comparison operations \{\( \neq, <, \leq, >, \geq \)\} can only be applied to atoms or atomic components of bitemporal atoms. The \( F \) may include \{\( \land, \lor, \neg \)\} as well.

**Cartesian Product** (\( \times \))  
Let \( R = < r(1), ..., r(n) > \) and \( S = < s(1), ..., s(m) > \). Then:

\[
E = R \times S \land E = < e(1), ..., e(n + m) >, \text{where}
\]

\[
E(i) = r(i) \text{ for } 1 \leq i \leq n
\]

\[
E(n + j) = s(j) \text{ for } 1 \leq j \leq m
\]

\[
E(R \times S) = \text{EV}(R) \times \text{EV}(S).
\]

**Unnesting** (\( \mu \))  
The unnesting operation makes a nested relation flatten. If attribute \( A \) of a bitemporal relation \( R \) is a set of atoms or bitemporal atoms, applying \( \mu \) to \( A \) will create a new tuple for each atom or bitemporal atom in the set of \( A \), and keep other
attributes unchanged. If $\mu$ is applied to every bitemporal set-valued attribute in $R$ recursively, then the final result will be a flat relation (1NF) with only atoms or bitemporal atoms, or nesting order of 0.

Let $R$ be a bitemporal relation with the schema $< r(1), ..., r(n) >$ and $< u(1), ..., u(m) >$ is the schema of $r(k), 1 \leq k \leq n$. Then:

$$E = \mu_k(R), \text{ where:}$$

$$e(i) = r(i) \text{ for } 1 \leq i \leq k - 1$$

$$e(i) = r(i + 1) \text{ for } k \leq i \leq n - 1$$

$$e(i) = r(i - n + 1) \text{ for } n \leq i \leq n + m - 1 \text{ and}$$

$$EV(E) =$$

$$\{ s[\exists \exists y(r \in EV(R) \land y \in r[k]) \land$$

$$s[i] = r[i] \text{ for } 1 \leq i \leq k - 1 \land$$

$$s[i] = r[i + 1] \text{ for } k \leq i \leq n - 1 \land$$

$$s[i] = r[i - n + 1] \text{ for } n \leq i \leq n + m - 1) \}$$

The new columns for the new tuple append to the end of $R$ for notational convenience.

**Nesting**(v) Nesting operation packs a bitemporal schema to make it more compact. If certain conditions are satisfied, nesting operations can reverse a zero order flattened bitemporal schema get from a nested bitemporal schema using unnesting operation.

Let $R$ be a bitemporal relation with the schema $< r(1), ..., r(n) >$ and $Y = < i_1, i_2, ..., i_k >$ is the set of $k$ attributes in $R$ where $0 \leq k \leq \deg(R)$ for some $k$, and $X$ is the set of remaining attributes in $R$, i.e., $\{1, ..., n - Y\}$. Let the arity of $X$ be $m$. Then:

$$E = \nu_{m+1-Y}(R) \text{ where}$$

$$e(j) = r(p) \text{ for } 1 \leq j \leq n - k, p \in X$$

$$e(n - k + 1) = \text{relation: } < r(i_1), ..., r(i_k) >$$

$$EV(E) =$$

$$\{ s[\exists r(y \in EV(R) \land$$

$$s[j] = r[p] \text{ for } 1 \leq j \leq n - k, p \in X \land$$

$$s[n - k + 1] = [z[\exists u(u \in E(R) \land u[p] = r[p] \text{ for } p \in X \land$$

$$z[j] = u[i_j] \text{ for } 1 \leq j \leq k])]\}$$

**Slice**(§) Let $R$ be a bitemporal relation, $k$ and $p$ are two of its attributes whose values are bitemporal atoms and $1 \leq k, p \leq n, k \neq p$ and $\theta \in \{\cup, \cap\}$, then:

$$EV(E) =$$

$$\{ s[\exists r(y \in EV(R) \land s[i] = r[j] \text{ for } 1 \leq i \leq n, i \neq k, i \neq p \land$$

$$s[k].V = r[k].V \land$$

$$s[k].TT = r[k].TT \land$$

53
\[ s[k].VT = r[k].VT \theta r[p].VT \wedge \]
\[ s[k].TT \neq \emptyset \wedge \]
\[ s[k].VT \neq \emptyset \]

Slice operation modifies the \( k^{th} \) attribute's bitemporal time component based on the temporal component of the \( p^{th} \) attribute by the operator \( \theta \in \{ \cup, \cap, - \} \). The result may not be empty. The \( \cap \) represents the "when" predicate in queries.

**Bitemporal Atom decomposition** (\( \delta \))  
Bitemporal atom decomposition operator splits the \( k^{th} \) attributes \( A \) of relation \( R \) into five new attributes that were it's components: \( A.TT_l, A.TT_u, A.VT_l, A.VT_u, A.V \). Let \( R \) be a bitemporal relation, \( k \) is a bitemporal attributes and \( 1 \leq k \leq n, t \leq now \). Then:

\[ E = \delta_k(R) \]
\[ e(i) = r(i) \] for \( 1 \leq i \leq k - 1 \)
\[ e(i) = r(i + 1) \] for \( k \leq i \leq n - 1 \)
\[ e(i) := \text{tuple < atom }> \] for \( n \leq i \leq n + 4 \)

\[ EV(E) = \]
\[ \{ s \mid \exists r \in EV(R) \wedge \}
\[ s[i] = r[i] \] for \( 1 \leq i \leq k - 1 \)
\[ s[i] = r[i + 1] \] for \( k \leq i \leq n - 1 \)
\[ s[n] = r[k].TT_l \wedge \]
\[ s[n + 1] = r[k].TT_u \wedge \]
\[ s[n + 2] = r[k].VT_l \wedge \]
\[ s[n + 3] = r[k].VT_u \wedge \]
\[ s[n + 4] = r[k].V \}

**Bitemporal Atom formation** (\( \tau \))  
This operation is the reverse of bitemporal atom decomposition (\( \delta \)). It combines existing attributes together to build a bitemporal atom. Let \( R \) be a bitemporal relation, \( k^{th}, l^{th}, m^{th}, q^{th} \) attributes are time attributes and \( p^{th} \) attribute is an atom. Then:

\[ E = \tau_{k,l,m,q,p}(R) \]
\[ r(i) := \text{tuple < atom }> \] for \( i = k, l, m, q, p \)
\[ e(i) = r(j) \] for \( 1 \leq i \leq n - 2, 1 \leq j \leq n, j \neq k, j \neq l, j \neq m, j \neq q, j \neq p \)
\[ e(n - 1) = \text{tuple :=< bitemporal atom >} \]

\[ EV(E) = \]
\[ \{ s \mid \exists r \in EV(R) \wedge \}
\[ s[i] = r[i] \] for \( 1 \leq i \leq n - 2, 1 \leq j \leq n, j \neq k, j \neq l, j \neq m, j \neq q, j \neq \wedge \]
\[ s[n - 1].TT_l = r[k] \wedge \]
s[n−1].TTu = r[l] ∧
s[n−1].VTl = r[m] ∧
s[n−1].VTu = r[q] ∧
s[n−1].V = r[p]}

Rollback(ρ) Let R be a bitemporal relation schema, <r(1),...,r(n)>, where r(k) is a bitemporal attribute. Let t be a time element and t < now. The rollback operator ρ is defined as:

\[ E_t = \rho_{t,k}(R) = \{s(\exists u)R(u) \land (s[i] = u[i] \text{ for } i = 1,...,n; i \neq k) \land s[k] = \{z(\exists x)(x \in u[k] \land (x.TTl \leq t < x.TTu \land z = k) \lor (z = x \land x.TTl < t \land \neg((\exists y)y \in u[k] \land y.VT \cap x.VT \neq \emptyset \land y.TTl \leq t < x.TTu))} \} \}

The result contains bitemporal atoms such that its TT component includes the time t. If we want to cut the transaction time of these bitemporal atoms so that y only includes time t, then we can replace in the above formula z = x by \( z = <[t, t + 1), x.TT, x.V) \).

4.3 Nested Bitemporal Relational Calculus

We discuss the relational calculus for a nested bitemporal relational model in this section[49].

Symbols Three types of symbols are defined for discussion in this section. 1) Predicate names are capitalized letters for representing relation schemas in a database, such as P, Q, R, S; 2) Variables are lower case letters for representing the tuple variables, such as s, t, u, v, etc. if s is a variable, then its \( i^{th} \) attribute will be denoted as \( s[i] \) for \( 1 \leq i \leq \text{deg}(s) \). Thus, if \( s[i] \) is a bitemporal atom, then \( s[i].TT, s[i].TTu, s[i].VTl, s[i].VTu, s[i].V \) represent the lower and upper bound of the transaction and valid time interval as well as the value component. 3) Constants are a, b, c, etc.
Well-Formed Formulas for Bitemporal Context  There are 5 atomic formulas and 3 formulas containing logical operators; these formulas are defined for bitemporal context as below:

1) $P(s)$: $P$ is a predicate name and $s$ is a variable.
2) $s[i] \text{ op } r[j]$, $s[i] \text{ op } c$, $c \text{ op } s[i]$ where $op \in \{=, \neq, <, \leq, >, \geq\}$ and $s[i]$, $r[j]$, $c$ are atomic.
3) $s[i].V \text{ op } p[j].V$, $s[i].V \text{ op } r[k]$, $s[i].V \text{ op } c$ where $op \in \{=, \neq, <, \leq, >, \geq\}$ and $s[i]$, $p[j]$, $r[k]$ are bitemporal atoms, $r[k]$ is an atom, $c$ is a constant. Besides the value component, $s[i]$’s temporal components $s[i].TT_I$, $s[i].TT_u$, $s[i].VT_I$, $s[i].VT_u$ are also allowed.
4) $s[i] \text{ op } r[j]$, $s[i] \text{ op } c$ where $op \in \{=, \neq\}$ and $s[i]$, $r[j]$ and $c$ have the same schema. If $s[i]$ and $r[j]$ are both temporal atoms, $s[i].VT \text{ op } r[j].VT$ is allowed.
5) Membership test:
   $s \in r[j]$ where $s$ is a variable or a constant, and $r[j]$ has the schema relation: $<s>$. $s \in c$ where $s$ is a variable or a constant, and $c$ is a constant, and they have the same schema.
6) $s[i] \in r[j]$ where $s$ is an indexed variable whose schema is an atom, and $r[j]$ is an indexed variable with the schema relation: $<u>$ and $u$ is an atom. If $s[i]$ is a bitemporal atom, then $u$, the tuple schema of indexed variable $r[j]$, is also a relation of bitemporal atoms.
7) $s[j] \in r[j].VT$ where $s[j]$ is an indexed variable whose schema is an atom and $r[j].VT$ is also an indexed variable representing a relation schema, which is a bitemporal atom.
8) if $\Psi$ and $\lambda$ are formulas, then we can have $\lambda \land \Psi$, $\lambda \lor \Psi$, and $\Psi \neg$.
9) if $\Psi$ is a formula with free variable $s$, then $\exists s \Psi(s)$ and $\forall s \Psi(s)$ are formulas and $s$ no longer occurs free in $\Psi$.
10) $r[j] = \{s[\Psi(s, u, v, ...)]\}$ is a formula with free variables $s$, $u$, and $v$. $r$ does not occur freely in $\Psi$.

Interpretation of Calculus Objects  The set $U$ is the universe of atoms. The interpretation of a nested bitemporal relational calculus object is relative to $U$. A derived set of $U$, denoted as $U^{ba}$, represents the domain of interpretation for a bitemporal atom. An interpretation of nested bitemporal relational calculus formula is an interpretation of each of its constants ($I_c$), predicate symbols ($I_p$), and assignment to each of its variables ($I_v$). If constant $c$ is an atom, then we have $Dom_c(U) = U$; if $c$ is a bitemporal atom, then $Dom_c(U) = U^{ba}$. $P$ is a predicate name, $I_p$ is a relation instance, and $I_p \in Dom_p(U)$. $s$ is a free variable, $I_s$ is a tuple instance and $I_s \in Dom_s(U)$. $I_s(i)$ denotes the $i^{th}$ component of $I_s$.

Formulas are interpreted as a boolean value: true or false. Rules for interpretation of formulas are as below:
4.4 SQL2011: Temporal Extension

SQL is the most widely used database query language standard. However, before SQL 2011's inclusion of temporal support, users had to implement the support for temporal data in their own application which made the development cycle expensive and complex. Along with the grow of the field, and discussions of possible solutions, the ISO SQL committee initiated a project to include a part in SQL for supporting temporal data in 1995; and released the extension in 2011. This subsection follows Kulkami and Michels discussion, and summaries the most important functionality of SQL2011 for supporting the temporal databases.

The most important part of the SQL:2011 extension for temporal data support is the ability to define and associate time periods simply with the rows of a table. Instead of introducing a new data type, Period, with an ordered pair of two time values, which is costly and hard for previous users to catch up, SQL:2011 adds Period definition as a metadata to tables. To deal the two different kinds of time, transaction time and valid time, SQL:2011 provides two types of Period: One is system-time period which handles transaction time, and application-time period which handles valid time. System-time period is defined by a specific name as SYSTEM_TIME while application-time period is
up to the user to give a name. For one table, at most one system-time and at most one application time period can be defined.

The operations for application-time period include insert, update, delete, etc. When a table contains application-time period is created, two columns are chosen to compose the valid time interval. Insertion will follow the definition of the period, for example, the format of time in the two columns. Update and delete operations use a syntactic extension, for portion of name-of-period, to specify the period of interest. For example, deleting/updating a portion of a period will result some sub periods to reflect the change.

The system-versioned tables are representing the transaction time of the record per the requirement of the application and for maintaining a history of the data. A system-versioned table should have a period definition with the standard-specified name, SYSTEM_TIME, and including the keywords WITH SYSTEM VERSIONING. Users can pick up any name for the start and end time value of the period. The facts that differ system-versioned tables with application-time tables are:

1. Users cannot change the value of the start and end columns. The value of the two columns will be generated automatically by the system.
2. INSERT will automatically setup the start time as current time when this action happens, and end value as the highest value of the column's data type. And if the end value is the highest value of the data type, we consider this record is the current system rows.
3. UPDATE and DELETE only operate on current system rows. Users cannot update or delete historical system rows.
4. UPDATE and DELETE will insert a historical system row when current system row is updated or deleted.

4.5 Connection Between Temporal Database and Semantic Web

As we already discussed, the concept of the Semantic Web and the widely used technology such as RDF triples, also the highly recommended Linked Data Initiative, are all trying to build a database that is decentralized, never out of date, and never run out of space. In this survey paper, we want to bring the temporal elements in database to the Semantic Web to make it more meaningful and useful.
4.5.1 Valid Time for RDF triples

RDF triples sometimes are statements without temporal attributes; and they are assumed to be true at present. However, we can add that attributes as a valid time attributes such as \([\text{date, present})\) to assert that fact. With that valid time interval, we can use RDF triple to describe some facts that were true in the past, or will be true in the future.

4.5.2 Transaction Time for RDF triples

RDF triples are lacking the timestamps right now for their simplicity. As we have already discussed, each RDF triple is a statement, which can have an inherent attribute as the timestamp when we share this statement over semantic web. Or, in database language if we regard the semantic web as a huge database, the timestamp of a RDF triple is the transaction time that we add this information into the database. With the timestamp, we are able to compare some statement that are describing the same fact, especially when they are conflict.

We examine currently used technologies in semantic web, and conclude with the following section.
5 Temporality in Semantic Web

5.1 Basic Expression

Current Semantic Web technologies have already included temporal elements, such as Date and time. For example, statement "William Web is a graduate student in 2016" has a temporal element of year. There are existing tags for RDF annotations that are used to express the temporal elements. The tags for expressing time is defined in RDF-compatible XSD types as below.

We also have tags defined in DCMI, such as date, dateAccepted etc as below:

We also have tags defined in Turtle, such as date, dateAccepted etc as below:

5.2 More Expressive

Furthermore, following the review of Ermolayev et al [19], current representation of temporal element for the semantic web is more than above tags.
OWL-time temporal ontology describes the temporal content and concepts including time points and intervals. However, this representation is different with the time representation in temporal databases. 1) OWL semantics is not equivalent to Entity Relationship model semantics; and 2) OWL syntax only provides binary relationships.

Among the various way of representing time for the semantic web, we are going to discuss Temporal Description Logic [36, 37], concrete domains [38], reification [39, 40], annotation [41], versioning [42], Named Graphs [42], and 4D fluents [44].

A summary of examples of these approaches is adopted from Ermolayev et al 2014 paper [19] as below:

Ermolayev et al also did a very good comparison of these various approaches:

### 5.2.1 Temporal Description Logic

Temporal description logics extends the standard description logics, which are formalisms designed for the semantic web [36, 47, 48]. Artale and Franconi introduced TDL, in particular, ALCQIT, through a detailed analysis of a case study [47]. This ALCQIT is obtained by combining a standard tense logic and the non-temporal description logic.
ALCQI with axioms [36]. Besides the basic types such as concepts, roles, and features in a concept language, tense-logical extensions are added such as future existential, past universal, etc as below.

However, Ermolayev et al [19] pointed out that the TDLs need extending OWL syntax and semantics by adding temporal constructs. And Artale and Franconi [36] admitted that representing information with temporal elements requires the support for con-
crete domains, thus, results in the proliferation of objects.

### 5.2.2 Concrete Domain

Following Baader and Hanschke 1991 paper for Concrete Domains [50], Carsten Lutz points out that Description Logics are closely related to modal logics, however, concrete domains is one of those means of expressivity that are not considered in modal logics. Concrete domains integrate concrete qualities - more specifically, time, numbers and strings - into DL concepts [38].

For example, "Expensive" is a concept that lack of concrete meaning, unless we integrate some numbers, "at least 20 euro". While "Expensive" becomes meaningful with this concrete domain, we can compare the concept "Expensive" by comparing the numbers. The other example is "WorkingTime" that needs a time interval to describe the execution time. Here we can integrate this concept with a concrete domain time intervals and their relationships [38].

Baader and Hanschke first formally revealed this issue and provided the solution as extending the description logic ALC with a concrete domain D to obtain ALC(D). As Lutz summarized, more precisely, ALC(D) is the extension of ALC by adding the ALC with:

- abstract features: roles interpreted as functional relations;
- concrete features: new type that is interpreted as a partial function from the logical domain into the concrete domain;

Then we get a new concept constructor that enables us to describe constraints on concrete values using predicates from the concrete domain [38].

### 5.3 Even More Expressive

However, as we are creating a database over the web, and we cannot ignore the temporal attributes of the data stored in the web. Some data has already included temporal prop-
erties. But most of time is only valid when we create them. After decades of creating the initial data and with the rapid change of the world, we need to compare different versions of the data, and conclude with most current or valid one. For example, "William is married to Nancy", which is recorded in 2012, however, after the divorce in 2015, this statement is not valid after 2015.

Thus, we need to attach a temporal attribute for each entity that it exists, each relationship that is valid, each statement that is made. As a result, if we are query about a fact, we can just query a slice of web that is valid for the time required in the query. For example, if in 2016 we create a query "Whom William is married to", the search will simply ignore the expired fact that "William is married to Nancy" which is valid in (2012-2015).

To add the temporal elements in Semantic Web, we need to add properties or statements in addition to current RDF model. One way of doing this, is the traditional reification process. We make additional statement about the Subject, Predicates, and Object so that the predicates have a valid time. Or, we can also consider the singleton property method, that create sub-properties for the original predicates.

6 Conclusion

Based on the widely use of computer and the Internet, Semantic Web is making the web as a database (see also linked data initiative). Instead of sharing databases as columns for certain attributes (some specific database), or as rows for certain entity (some wiki-data page for a country), Semantic Web, especially the RDF triple approach, shares data by cells. Each RDF triple statement represents a simple fact that tells a piece of information.

Time ontology provides a good base for us to use temporal elements. RDF and its annotations cover a part of temporal elements in their syntax so we can express them and use them as useful data. However, it is still not enough if we want to describe and express the temporal attributes for the statements themselves.

Temporal Semantic Web makes data more useful and valid. By introducing the transaction time and valid time difference, current data over the Semantic Web can be more meaningful and reliable. Along with the explosion of information, we can get the most valid and accurate information. However, there are still many works and challenges,
both the effective and efficiency of implementing the temporal logic into the RDF languages.
References


67


[54] C. Atay; A. Tansel (2009). Bitemporal Databases Modeling and implementation. VDM,