Annotated Schema (1.1) : Mapping Ontologies onto Dataset Schemas (2023/09/28)
Bradley Huffaker and and kc claffy

– Goals –

The goal of this proposal, Annotated Schema (AS), is to provide a limited ontology of annotations for dataset metadata that inform a prospective user of the classes, properties, and identifiers contained in the data. The intended audience of this document includes Data Curators (annotators) who want to create meaningful dataset descriptions, and those searching for data sets in our catalog.

What is an ontology? Ontologies attempt to create an encompassing data representation, formal naming, and definition through the abstraction of **individuals** (objects), **classes** (sets, concepts, types of objects, or kinds of things), **properties** (aspects, attributes, characteristics), **relationships** (ways that classes and individuals relate to one another), and **restrictions** (formal relationships required for an assertion about the classes or individuals to be true).

Our guiding principle is to be descriptive, not prescriptive. That is, the annotated schema uses annotations to describe messy real world datasets to aid discovery and high level understanding. It is not a goal of this annotated schema to force the datasets to match a strict ontology or schema, but to help researchers make their own decisions about representations and curation that meet their needs. **To ease the burden on the Data Curator, this proposal includes annotations for classes and properties, but does not include machine-readable relationships or restriction**s. Beyond the scope of this proposal are issues related to annotating the dataset's actual data, validating the data, or transforming the data into Resource Description Framework (RDF) *triples*.[1]

Distinguishing our Goals from those of Related Systems.

The goal of projects such as Google's Knowledge Graph [GKG] and W3C's OWL's [OWL] semantic web is to provide a human and machine-readable data representation, embed data into that representation, and use that embedded data for machine learning and querying. This proposal attempts only the first of these goals, with a primary focus on human understanding,

[1] Resource Descriptions Framework is a web framework used to represent interconnected data on the web.

Bradley Huffaker University of California San Diego, US
kc claffy University of California San Diego, US

: bhuffake@beanball:/project/geo-asn-ranking/20230301; vim
This material is based on research sponsored by the National Science Foundation (NSF) grant OAC-2131987. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF.

and a secondary focus on machine readability only of the dataset's metadata. In contrast, the RDB to RDF Mapping Language (R2RML) [R2RML] supports automated mapping from relational datasets to RDF *triples*. To enable this automation, R2RML requires the Data Curator to explicitly express all edge cases in a programmatic syntax (or filter those cases before the transformation). This requirement greatly increases complexity, requiring the Data Curator to make decisions on behalf of future users of the data. In contrast, our approach prioritizes informing the consumer so they can make their own choices.

In addition to providing the ability to create informative annotations, our proposed Annotated Schema will provide useful dataset metadata as structured data such as that consumed by Google. Google's Data Search [GO-DATA] supports Schema.org's Dataset [SCHEMAORG-DAT] and W3C's Comma Separated Values on the Web (CSVW) [CSVW]. Schema.org is designed to provide richer annotation of webpage content. In 2013, Google's Benjello Jacob proposed adding dataset table header annotations to Schema.org's vocabulary on the public-vocabs@w3.org mailing list [PUBLIC-VOCABS]. This thread resulted in a new proposal: "Describing tables with schema.org" [SCHEMAORG-TAB], which as far as we can tell failed to get traction. In the following year, Schema.org discussions moved to its GitHub repository's issues [SCHEMAORG-IS]. The last issue to reference Benjello Jacob's original email thread closed in 2018 [SCHEMAORG-ISS]. This thread introduced a new property to schema.org's Dataset class: the *variableMeasured* property, which describes the type of variable (e.g., integer, string) represented in a column. But a gap remains: the ability to identify columns with classes or class properties.

Perhaps as a consequence of this gap, Google added beta support for dataset table annotations using the CVSW format [GO-DATA-CSVW]. Similar to R2RML, CSVW is designed to provide CSV data annotations, with a focus on automated machine conversion into RDF triples. CSVW supports cell and column property annotations, but does not support class annotations. Our Annotated Schema proposal addresses this limitation with the introduction of *Entities*.

– Overview of Proposed Annotation Schema –

Our proposal uses Schema.org's Dataset type and W3C's CSVW class as a starting point. We encode the dataset's metadata in Schema.org's *Dataset* class.[2] We encode column header information in CSVW format. We add a new *Entity* class which will encode the mapping between the columns and classes, properties, and individuals.

– Relational Tables –

We represent each dataset using a set of relational tables. Tables are typically a good representation of our datasets, and provide a compact human-friendly visual representation. Real world datasets come in many forms, including APIs and flat files. A single file may contain multiple tables, and a table can span multiple files. The goal of our Annotated Schema is to represent each dataset as clearly as possible.

[2] Schema.org uses *type* and *class* interchangeably; we will use *class* in this document.

For illustration purposes, in the remainder of this tutorial document we consider a dataset that we represent as three tables:

- The top table represents information about the company's employees.
- The bottom right table represents information about the company's projects,
- The bottom left table shows which employees (identified by their SSN) work on which project.

Note that the first column of the bottom left table has the column name "*name"*. This column name is not shared with a column name in the other two tables. A better choice would have been "*project name*". Real world datasets can have such poor naming choices. Part of our motivation for this Annotated Schema proposal is to create better annotations (than real world datasets may have) to improve clarity and facilitate discovery of data sets.

In the dark gray headers are the dataset's original column headers and table names. The light gray headers are the first set of annotations. We will incrementally add new colored headers to these tables as we introduce new layers of annotations.

Table	**name**	Employee							
	titles	**Cool Company Employees**							
Column	**name**	last_name	first_name	ssn	pay	m_last_name	m_first_name	address	home_state
	titles	**last name**	**first name**	**SSN**	**pay**	**manager last name**	**manager first name**	**address**	**home state**
	example	Smith	Tom	555-555-5555	90K	Sue	Mary	888 Cats Str San Diego	CA

Table 1.a: *Employee Table*

Table	**name**	Project/Employee	
	titles	**Project's Employees**	
Column	**name**	project	ssn
	titles	**name**	**ssn**
	titles	Big Splash	555-555-5555

Table 1.b: *Project/Employee Table*

Project		
Cool Company Projects		
project	budget	description
project name	**budget**	**description**
Big Splash	900M	Make a big splash

Table 1.c: *Project Table*

To lower the burden of manual annotation, the Annotated Schema will also support a YAML [YAML] representation of the JSON-LD [JSON-LD] encoding. The first *document*[3] in the YAML format must be of Schema.org *Dataset* type and will contain the dataset's metadata. Subsequent documents must be W3C CSVW classes with an additional *Entity* class that we (CAIDA) created. If the Data Curator represents the dataset as multiple tables, CAIDA's annotation software that converts from YAML to JSON-LD will use the CSVW class *tableGroup*

[3] YAML format supports multiple *documents* in the same file delimited by "---", the first of these documents must contain the dataset metadata. The following documents represent the individual table metadata.

to group the tables (illustrated below).

YAML representation

```
---
# Dataset metadata
"@context": "https://schema.org/"
"@type": Dataset
---
# First Table metadata
"@context":
  - "https://www.w3.org/ns/csvw#"
  - "schema": "http://schema.org/"
"@type": Table
"schema:title": "Cool Company Employees"
tableSchema:
  columns:
    - name: ssn
      titles: SSN
      datatype: string
      cells:
        - value: 555-55-5555

---
# Second Table
"@context":
  - "https://www.w3.org/ns/csvw#"
"@type": Table
"schema:title": "Cool Company Projects"
```

Structured Data JSON-LD

```
{
  "@context": [
    "https://schema.org/",
    {"csvw": "https://www.w3.org/ns/csvw#"}
  ],
  "@type" "Dataset",
  "name": "Cool Company DB",
  "mainEntity" : {
    "@type":"csvw:tableGroup": {
    "csvw:tables": [
    {
      "@type": "csvw:Table",
      "schema:title": "Cool Company Employees",
      "csvw:tableSchema": {
      "csvw:columns": [
      {
        "csvw:name": "ssn",
        "csvw:titles": "SSN",
        "csvw:datatype": "string",
        "csvw:cells": [ {
           "csvw:value": "555-55-5555"
        } ]
       },
```

illustrating conversion from YAML to JSON-LD representation of data.

A column's name provides a unique identifier for the column, while a column's titles (may be more than one title) provide a list of possible headers for the column. JSON-LD also supports single and list values for the same property, such as the below example:

`"titles": "IPv4"`	`"titles": ["IPv4", "Internet Address version 4"]`

illustrating the same property have a single or array as a value

CVSW is uninterested in the presentation of the tables it encodes. As a result, it does not provide any column presentation functionality. In order to encode this information for use with CAIDA's catalog Web interface, Annotated Schema has added the ***columnReferences*** property to CSVW's *column*. This property has a list of other child columns which are nested under the parent in a visual representation.

```
columns:
 - name: name
   title: Name
   dateType: string
 - name: "salary>month"
```

Name	Salary	
	Monthly	Yearly
smith	2000	24000

```
  title: Month
  dataType: integer
- name: “salary>year”
  title: Month
  dataType: integer
- name: salary
  title: Salary
  dataType: object
  "caida:columnReferences": [ "salary>month",
"salary>year" ]
```

– Entity –

The Data Curator can group columns in a table such that the values in a single row describe properties about a single individual. For example, in the “*employee*” table, the first four columns of the first row (“*last name”, “first name”, “SSN”* and *“pay”*) describe properties of an individual employee, e.g., Smith. The next two columns (“*manager last name”* and “*manager first name”*) describe properties about the employee’s manager (e.g., Sue). Individuals described by the same set of columns have a shared implied property embedded in the structure of the table. For example, in Table 1(a) the first set of four columns describe individuals who are employees, the next two columns describe managers, and the last two (“*home state”* and “*address”*) describe the employee home addresses. We introduce the term ***Entity*** to describe this kind of potentially multi-column property.

In our example, we have four entities: *“employee”, “manager”, “employee address”, and “project”.* In Table 4.a we have annotated these with a new *Entity* row. Columns that share the same entity annotation have the same color and name in the *Entity* row. In the employee table (Table 4.a), while an individual could theoretically be their own manager, in general these are two different individuals. Providing different entity names for the employee and manager facilitates understanding of the table.

Table	**name**	Employee							
	titles	**Cool Company Employees**							
Entity	**name**	employee				manager		employee address	
Column	**name**	last_name	first_name	ssn	pay	m_last_name	m_first_name	address	home_state
	titles	**last name**	**first name**	**SSN**	**pay**	**manager last name**	**manager first name**	**address**	**home state**
	example	Smith	Tom	555-555-5555	90K	Sue	Mary	888 Cats Str San Diego	CA

Table 2.a: *Employee Table*

Entity	name	project	employee
Column	name	name	ssn
	titles	Big Splash	555-555-5555

Table 2.b: *Project/Employee Table*

project		
project name	budget	description
Big Splash	900M	Make a big splash

Table 2.c: *Project Table*

In the example below, we added CAIDA's *Entity* class and *entities* property. The added *caida* context is in red, the *entities* property is in blue, and the Entity class (aka type) is in green.

```
"@context":
  - "https://www.w3.org/ns/csvw#"
  - "schema": "http://schema.org/"
  - "caida": "http://catalog.caida.org/ontology/"
tableSchema:
  "caida:entities":
    - "@type": "caida:Entity"           <- CAIDA Entity class/type
      "caida:name": employee
      "caida:columnReferences": ["last_name", "first_name", "ssn", "pay"]
```

– Classes –

A Data Curator can group these entities into ***classes***. The example shown in the table below has three classes: *Person*, *Address*, and *Project*. *Person* is the class for both the employee and manager entities. Table 3 adds a *class* annotation row to the tables to provide a class identifier for each entity.

The same column may map to multiple class annotations. For this example, the column "*home_state*" can map to two classes: *Address* and *Region*. A class need not use all the information in a column. For example, the *City* class would only use the string "San Diego" in the *address* column.

Table	name	Employee							
	titles	Cool Company Employees							
Entity	name	employee				manager		employee address	
	class	Person				Person		Address	
Column	name	last_name	first_name	ssn	pay	m_last_name	m_first_name	address	home_state
	titles	last name	first name	SSN	pay	manager last name	manager first name	address	home state
	example	Smith	Tom	555-55 5-5555	90K	Sue	Mary	888 Cats Str San Diego	CA

Table 3: *Adding class annotation row.*

Below we have added the property *class* to CAIDA's Entity class to allow curators to specify the entity's class property. A good resource for Data Curators looking to find existing classes and properties to use in creating annotations is the Linked Open Vocabularies database (https://lov.linkeddata.es/dataset/lov) or CAIDA's ontology (https://catalog.caida.org/ontology).

```
tableSchema:
  entities:
    - "@type": "caida:Entity"
      "caida:name": employee
      "caida:classUrl": "schema:Person"
      "caida:columnReferences": ["last_name", "first_name", "ssn", "pay"]
```

CAIDA's catalog will use the following Schema.org Class properties: *name*, *alternativeName*, *description*, and *url* [SCHEMAORG]. Many web pages in the catalog have limited available display space, so we have added a *label* property for UI purposes. If the *label* is not provided, CAIDA's conversion (from YAML to JSON-LD) tool will set *label* to the shorter of *name* or *alternativeName*. If neither *name* nor *alternativeName* is provided, the conversion tools will use the last word in the class' URI (so in the case of http://xmlns.com/foaf/0.1/Person the conversion tool will use *Person*). If two classes have the same *label*, the conversion tool will prepend the *term* defined in the context (above "foaf") to the *label* (resulting in *foaf:Person*)

Note: Multiple annotation variants are possible. For example, below the two columns "*address"* and "*home state"* are grouped together into class *Address*, but the Data Curator could have annotated them separately with classes *Street* and *Region*. Additionally, a column can be shared across multiple entities. In the right table below, the column "*address*" is shared by both the entity *City* and *Address*. The Data Curator should use the combination that maximizes clarity.

employee address	
Address	
address	**home state**
888 Cats Str San Diego	CA

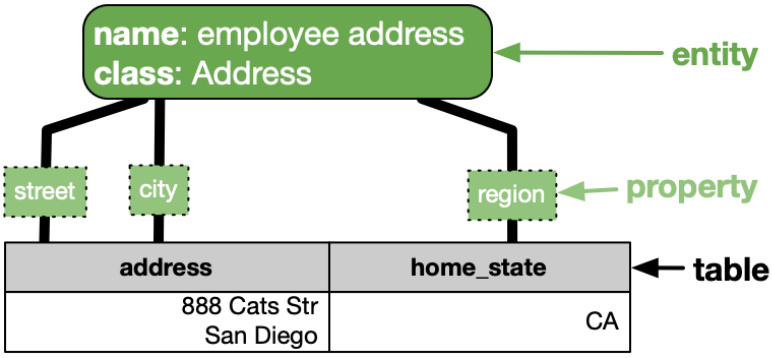
employee address		
Address		
	employee city	employee region
	City	Region
address		**home state**
888 Cats Str San Diego		CA

– PropertyMaps –

Properties are aspects, attributes, or characteristics of classes. The Data Curator can map one or more table columns to a property. A ***PropertyMap*** provides the mapping between a property

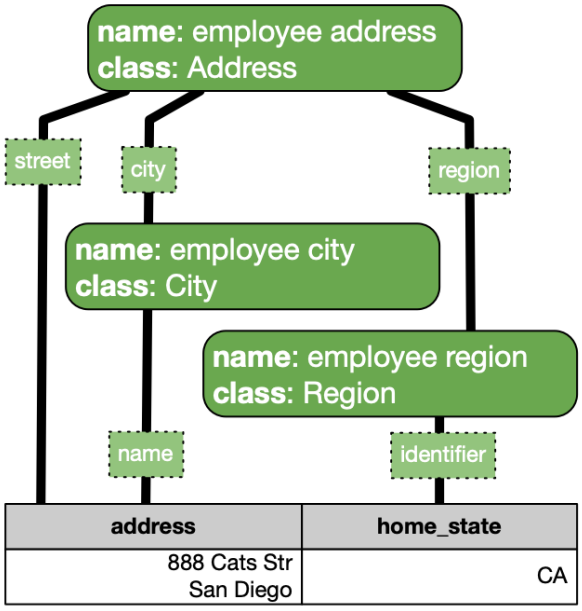
and a list of columns. Each *PropertyMap* includes two properties: *property* and *columns*. (Yes, the property is a property!) In the example below the "*employee address*" entity's *propertyMaps* contains a mapping from the properties *street* and *city* to the column "*address*", while the "*employee address*" entity's *region* property maps to the column "*home_state.*"

Note: Classes start with uppercase letters and properties start with lowercase letters. So *Table* is a class, while *table* is a property.

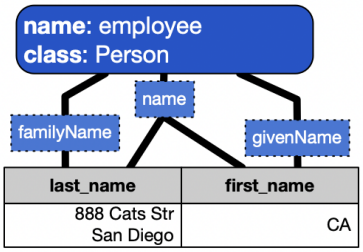


PropertyMaps	
property	**columns**
employee address	
street	address
city	address
region	home_state

A Data Curator may define properties in terms of properties of a declared child entity. This nesting provides a more detailed description of the classes and properties contained in the table. To enable this nesting, a PropertyMap would include the child entity's names in the PropertyMap's *entities* property. The parent entity's *PropertyMap* maps the property to the child *entity*. The child *entity's PropertyMap* maps the property of the child entity's class to columns. In the example *PropertyMaps* below, the parent entity "*employee address"*s property "*city"* maps to the child entity "*employee city"*, which has its own property map. In the "*employee city*"'s PropertyMap, the property *name* maps to the column "*address*".



PropertyMaps		
property	**entities**	**columns**
employee address		
street		address
city	employee city	
region	employee region	
employee city		
name		address
employee region		
identifier		home_state



PropertyMaps		
property	entities	columns
employee region		
familyName		last_name
givenName		first_name
name		last_name, first_name

Some properties combine multiple columns. In the example below the property *name* contains the columns "*last_name*" and "*first_name*". In other cases the Data Curator may wish to define the property as a combination of other properties such as "*street_city_region*". Properties do not always map one-to-one onto a column, so property hierarchies allow a Data Curator to specify that the "*street_city_region*" is using the same values as "*street*", "*city*", and "*region*". The following table contains the added *PropertyMap* property.

Table	name	Employee									
	titles	Cool Company Employees									
Entity	name	employee				manager		employee address			emp..region
	class	Person				Person		Address			Region
PropertyMap	property	name		identity number	salary	name		steet_city_region			identifier
		familyName	givenName			familyName	giveName	street	city	region	
	entities								emp..city	emp..region	
	columns	last_name	first_name	ssn	pay	m_last_name	m_first_name	address			home_state
		last_name, first_name				m_last_name, m_first_name					
Column	titles	last name	first name	SSN	pay	manager last name	manager first name	address		home state	
	example	Smith	Tom	555-555-5555	90K	Sue	Mary	888 Cats Str San Diego		CA	

Table 4: *We have added the propertyMaps property to the Entity to store a list of PropertyMaps. The PropertyMap (the set of rows in the red dashed line) has four properties: propertyUrl, columnReference, and entityReference.* The properties columnReference and entityReference can store one or a list.

```
- "@type": "caida:Entity"
   "caida:name": employee
   "caida:classUrl": "schema:Person"
```

```
    "caida:columnReferences": ["last_name", "first_name", "ssn", "pay"]
    "caida:propertyMaps":
     - "caida:propertyUrl": "schema:familyName"
       "caida:columnReferences": last_name"
     - "caida:propertyUrl": "schema:givenName"
       "caida:columnReferences": given_name"
     - "caida:propertyUrl": "schema:name"
       "caida:columnReferences": ["last_name", "first_name"]
  - "@type": "caida:Entity"
    "caida:name": Employee region
    "caida:classUrl": "schema:State"
    "caida:columnReferences": "home_state"
    "caida:propertyMaps":
     - "caida:propertyUrl": "schema:identifier"
       "caida:columnReferences": "home state"
   - "@type": "caida:Entity"
    "caida:name": Employee Address
    "caida:classUrl": "caida:Address"
    "caida:columnReferences": ["address", "home_state"]
    "caida:propertyMaps":
     - "caida:propertyUrl": "caida:street"
       "caida:columnReferences": "address"
     - "caida:propertyUrl": "caida:city"
       "caida:entityReference": "Employee City"
     - "caida:propertyUrl": "caida:region"
       "caida:entityReference": "Employee Region"
```

CSVW only supports *propertyUrl* for a single column, so when our conversion tools create Google-friendly structured data, these tools will copy over only the *propertyUrl* when the *PropertyMap* contains a single column, illustrated below.

```
- "@type": "caida:Entity"
  "caida:name": employee region
  "caida:classURL": "http://...#Region"
  "caida:columns": "home_state"
  "caida:propertyMaps":
    - "caida:propertyUrl": "http://...#identifier"
      "caida:columnReferences": "home_state"
```

```
"csvw:columns": [
{
   "csvw:name": "home_state",
   "csvw:titles": "home state",
   "csvw:datatype": "string",
   "csvw:propertyUrl:": "http://...#identifier"
},
```

– Namespace –

A Data Curator can specify one or more ***Namespaces*** (a class, so capitalized) as part of a *PropertyMap*. A Namespace provides a guide to the set of class identifiers contained in the

property. In our example, the "*employee*" entity's "*identity number*" property has identifiers from the U.S. Social Security Number namespace, and "*employee region*" entity's "*identifier*" has identifiers from ISO 3166 country codes.

Namespaces are descriptive, not prescriptive. A dataset that mostly uses the IATA airport code for cities, with a few exceptions, could be annotated with the IATA airport code namespace. A property may have multiple Namespaces. A "*national identification number"* property could include two Namespaces: U.S. Social Security Numbers and Canada's Social Insurance Numbers.

Table	**name**	Employee										
	titles	**Cool Company Employees**										
Entity	**name**	employee				manager		employee address			emp..region	
	class	Person				Person		Address			Region	
PropertyMap	**property**	name		identity number	salary	name		steet_city_region			identifier	
		familyName	givenName			familyName	giveName	street	city	region		
	entities								emp..city	emp..region		
	namespaces			us-ssn				us-postal-address			iso-alpha2	
	columns	last_name	first_name	ssn	pay	m_last_name	m_first_name	address			home_state	
		last_name, first_name				m_last_name, m_first_name						
Column	**titles**	**last name**	**first name**	**SSN**	**pay**	**manager last name**	**manager first name**	**address**		**home state**		
	example	Smith	Tom	555-555-5555	90K	Sue	Mary	888 Cats Str San Diego		CA		

Table 5: *We have added the namespaces property to the PropertyMap which contains a list of Namespaces.*

CAIDA's catalog uses the *Namespace's name*, label, *description*, and url.

```
"caida:propertyMaps":
 - "caida:columns": "ssn"
   "caida:propertyUrl": "https://w3id.org/sbeo#id"
   "caida:namespaceUrl": "caida:UsaSsnNamespace"
```

– SubjectEntity –

Many tables have a primary subject entity (***subjectEntity***). For example, the "*Employee*" table mostly describes the employees, and not the managers or employee addresses. Thus the "*employee*" entity is the *subjectEntity* of the table. *SubjectEntities* are optional; not all tables will

have a good subject. If the table contains a single entity it will be the table's subject entity.

The Data Curator should identify the *subjectEntity* of a table (if there is one) that provides the greatest clarity. In our example, the "*Employee*" Table's subject is the "*Employee*" and the "*Project*" Table's subject is the "*project*". The "*Project's Employees*" Table (Table 6b) could have a subject of "*project*", "*employee*", or none. If the Data Curator wants to suggest that the subject of the dataset as a whole is the employee, then they could annotate the table's subject entity as "employee". Table 6 adds a new **subjectEntity** row.

Table	name	Employee									
	titles	Cool Company Employees									
	subjectEntity										
Entity	name	employee				manager		employee address			emp..region
	class	Person				Person		Address			Region
PropertyMap	property	name		identity number	salary	name		steet_city_region			identifier
		familyName	givenName			familyName	giveName	street	city	region	
	entities								emp..city	emp..region	
	namespaces			us-ssn				us-postal-address			iso-alpha2
	columns	last_name	first_name	ssn	pay	m_last_name	m_first_name	address			home_state
		last_name, first_name				m_last_name, m_first_name					
Column	titles	last name	first name	SSN	pay	manager last name	manager first name	address			home state
	example	Smith	Tom	555-555-5555	90K	Sue	Mary	888 Cats Str San Diego			CA

Table 6.a: *Employee with a new subjectEntity row*

table	name	Project's Employees	
	subjectEntity		
Entity	name	project	employee
	PropertyMap	...	...
Column	titles	name	ssn
	example	Big Splash	555-555-5555

Table 6.b: *Project/Employee Table*

Cool Company Projects		
project		
project		
...		
project name	budget	description
Big Splash	900M	Make a big splash

Table 6.c: *Project Table*

Similarly, we have added a ***subjectEntityReference*** field to the Table description below.

```
"@type": Table
"dc:title": "Cool Company Employees"
```

```
"dcat:keyword": ["employee", "pay"]
"caida:subjectEntityReference": "employee"
```

– YAML short cuts –

A Data Curator may manually or programmatically create the JSON-LD directly, but to lower the burden of generating manual YAML annotations we will support the following short cuts.

#Documents: YAML supports storing the Dataset and Tables in the same file as separate *documents* using the "---" YAML marker. The conversion script will compile them into JSON-LD.

```
---
# Dataset metadata
"@context": "https://schema.org"
"@type": Dataset

---
# First Table metadata
"@context":
  - "https://www.w3.org/ns/csvw#"
  - "dc": "http://purl.org/dc/terms/"
"@type": Table
```

#CopyDatasetUrl: A dataset may have almost exactly the same metadata as an existing dataset. Consider two packet traces taken on different dates. The second dataset's metadata is identical to the first except for the date. To support this scenario, our Annotated Schema specification includes the **CopyDatasetUrl** command in the YAML representation.

```
---
# Dataset metadata
"#copyDatasetUrl": "https://catalog.caida.org/dataset/telescope_darknet_scanners"
"dateCreated": "2023/04/23"
```

#CopyTableUrl: This commend is the same as CopyDatasetUrl, but copies over the content of a single table. The Data Curator specifies the table name after the '#' anchor in the URL.

```
---
# First Table metadata
"#copyTableUrl": "https://catalog.caida.org/dataset/as_rank#as_information"
```

caida: To reduce the syntactic burden, the Data Curator may omit the "caida:" for the CAIDA Entity related properties, and the conversion script will add it to the compiled JSON-LD

representation.

```
tableSchema:
  "caida:entities":
    - "@type": Entity
      name: employee
      classUrl: "http://xmlns.com/foaf/0.1/Person"
      columns: ["last_name", "first_name", "ssn", "pay"]
```

example: To reduce the syntactic burden of data curatoin, we will support the "*example*" property on a Table Column. The conversation script will map this syntax into the "cell":["value":"XXX"] JSON-LD representation.

```
columns:
  - name: ssn
    titles: SSN
    datatype: string
    example: 555-55-5555
```

```
columns:
  - name: ssn
    titles: SSN
    datatype: string
    cell:
      -value: 555-55-5555
```

– Review –

We will be representing metadata for CAIDA's datasets using a combination of Schema.org's Datasets (schema.org/Dataset) and W2C's CSVW tables (www.w3.org/ns/csvw). We model our approach on Google's Dataset's Structured Data standard (developers.google.com/.../dataset), with the addition of Entities to CSVW's tables.

Entities allow us to differentiate individuals of the same class annotated in the same row, and express more complex mappings between property and columns. We add to CSVW's tables the property **entities** as a list of Entities represented in the table, and **subjectEntity** to contain the name of the entity that is the subject of the table.

Entity is a logical set of table columns annotating the same individual. An **Entity** contains the following properties: **name** which uniquely identifies the Entity in the table, **classUrl** which identifies the Entity's class, **columns** which contains a list of the column names that describe entity values, and **properties** with a list of PropertyMaps.

PropertyMap is a mapping between class properties and table columns. A **PropertyMap** contains a **propertyUrl** with a URL pointing to a property definition, and optional **columns** which contains the set of columns needed to create the property value, **entities** which contains the set of child entities, and namespaceUrl defining an identifier namespace for the property's values.

We propose to use this standard to provide annotations for additional objects such as papers,

presentations, recipes, or software.

General class declarations.

References:

[GKG] “How Google's Knowledge Graph works - Knowledge Panel Help.” *Google Support*, https://support.google.com/knowledgepanel/answer/9787176?hl=en. Accessed 22 February 2023.

[OWL] “Web Ontology Language (OWL)”, W3C, https://www.w3.org/OWL/. Accessed 23 February 2023.

[RDF] “Resource Description Framework (RDF) ”, W3C, https://www.w3.org/RDF/. Accessed 23 February 2023.

[R2RML] “R2RML: RDB to RDF Mapping Language”, https://www.w3.org/TR/r2rml/ accessed 10 March 2023

[RML] “RDF Mapping Language”, W3C, https://rml.io/specs/rml, Accessed 11 March 2023

[CSVW] “CSVW Nampespace Vocabulary Terms - W3C”, W3C, https://www.w3.org/ns/csvw, accessed 10 March 2023

[JSON-S] “JSON Schema | The home of JSON Schema”, “http://json-schema.org”. JSON-Schema, Accessed 24 February 2023.

[JSON-LD] “JSON for Linking Data”, “https://json-ld.org/”, Accessed 21 February

[YAML] “YAML Ain’t Markup Language”, “https://yaml.org/”, Accessed 21, February

[Semantic] “Semantic network”, “https://en.wikipedia.org/wiki/Semantic_network” , Accessed 24 February 2023

[SEM-DEAD] “The semantic web is dead”, https://terminusdb.com/blog/the-semantic-web-is-dead/, Accessed 21 February 2023

[IP-COM] “IP Networks Topology an Communications Ontology”, https://github.com/twosixlabs/icas-ontology/blob/master/ontology/ipnet.ttl, Accessed 24 February 2023

[SCHEMAORG] “schema.org”, https://schema.org/, accessed 10 March 2023

[SCHEMAORG-DAT] “Dataset a Schema.org Type”, https://schema.org/Dataset, accessed 10 March 2023

[PUBLIC-VOCABS] https://lists.w3.org/Archives/Public/public-vocabs/2013Aug/0033.html, 13 Aug 2013

[SCHEMAORG-TAB] “Describing tables with schema.org”, https://lists.w3.org/Archives/Public/public-vocabs/2013Aug/att-0033/Lookinginsidetables.html, Accessed 10 March 2023, last updated 8 May 2013

[SCHEMAORG-IS] “Improving Dataset descriptions”, https://github.com/schemaorg/schemaorg/issues/1083, accessed 10 March 2023, issue closed 2018, last comment 2020

[GO-DATA] “Dataset (Dataset, DataCatalog, DataDownload) structured data”, https://developers.google.com/search/docs/appearance/structured-data/dataset, Google, accessed 11 March 2023

[GO-DATA-CSVW] “Dataset (Tabular dataset)”, https://developers.google.com/search/docs/appearance/structured-data/dataset#

tabular, Google, accessed 11 March 2023
[SCHEMAORG-IS] https://www.w3.org/wiki/WebSchemas/SchemaDotOrgProposals last updated 12 May 2015