Managing linked open data in Wikidata using the W3C *Generating RDF from Tabular Data on the Web* Recommendation

Steven J. Baskauf[a,*] and Jessica K. Baskauf[b]
[a] *Jean and Alexander Heard Libraries, Vanderbilt University, Nashville, Tennessee, USA*
E-mail: steve.baskauf@vanderbilt.edu, https://orcid.org/0000-0003-4365-3135
[b] *Carleton College, Northfield, Minnesota, USA*[1]
https://orcid.org/0000-0002-1772-1045

Abstract. The W3C *Generating RDF from Tabular Data on the Web* Recommendation provides a mechanism for mapping CSV-formatted data to any RDF graph model. Since the Wikibase data model on which Wikidata is built can be expressed as RDF, this Recommendation can be used to document tabular snapshots of parts of the Wikidata knowledge graph in a simple form that is easy for humans and applications to read. Those snapshots can be used to document how subgraphs of Wikidata have changed over time and can be compared with the current state of Wikidata using its Query Service to detect vandalism and value added through community contributions.

Keywords: CSV file, Wikibase model, SPARQL

1. Introduction

Because of its availability and ease of use, Wikidata has become one of the widely used open knowledge graphs [1]. Its dedicated users and easy-to-use graphical interface promise value added through community contributions, and access through its API[2] makes it possible for large amounts of data to be contributed via bot scripts [2] that upload data from a variety of sources.

These advantages have generated a lot of interest in communities such as galleries, libraries, archives and museums (GLAM) [3], biodiversity informatics [4], and information specialists [5] for using Wikidata as a place to expose and manage data about items of their concern, such as collections records, authors, and authority files. However, since anyone can edit Wikidata, these communities are also concerned with monitoring statements about those items to revert vandalism as well as to discover when contributions are made by contributors outside their community.

The Wikibase model. Wikidata is built on a general model known as the Wikibase data model [6]. The Wikibase data model has an RDF export format [7] whose key components are shown in Fig. 1. The primary resource of interest in the model is some describable entity known as an *item*. One can describe an item directly using a "truthy" statement -- an RDF triple where the predicate has the `wdt:` prefix (Table 1) and the object is a simple value literal or IRI. Statements about items are also effectively reified by instantiating a *statement* node. The statement node serves as the subject of other triples that link to *references* that document the statement, *qualifiers* that provide context to the statement, and *value nodes* that make it possible to describe complex values like time, quantities, and globecoordinates.

* Corresponding author
[1] Now at Google
[2] https://www.wikidata.org/w/api.php

Table 1
Namespace abbreviations

Prefix	Namespace IRI
rdfs:	http://www.w3.org/2000/01/rdf-schema#
prov:	http://www.w3.org/ns/prov#
schema:	http://schema.org/
wd:	http://www.wikidata.org/entity/
wds:	http://www.wikidata.org/entity/statement/
wdt:	http://www.wikidata.org/prop/direct/
p:	http://www.wikidata.org/prop/
ps:	http://www.wikidata.org/prop/statement/
psv:	http://www.wikidata.org/prop/statement/value/
pq:	http://www.wikidata.org/prop/qualifier/
pqv:	http://www.wikidata.org/prop/qualifier/value/
pr:	http://www.wikidata.org/prop/reference/
prv:	http://www.wikidata.org/prop/reference/value/
wikibase:	http://wikiba.se/ontology#

Items also have language-tagged *labels* and *descriptions* with a maximum of one of each per language, and an unlimited number of *aliases* per language.

Advantages of tabular data. Because the Wikibase data model that underlies Wikidata can be expressed as RDF, subgraphs of the Wikidata knowledge graph can be exported in one of the RDF serializations

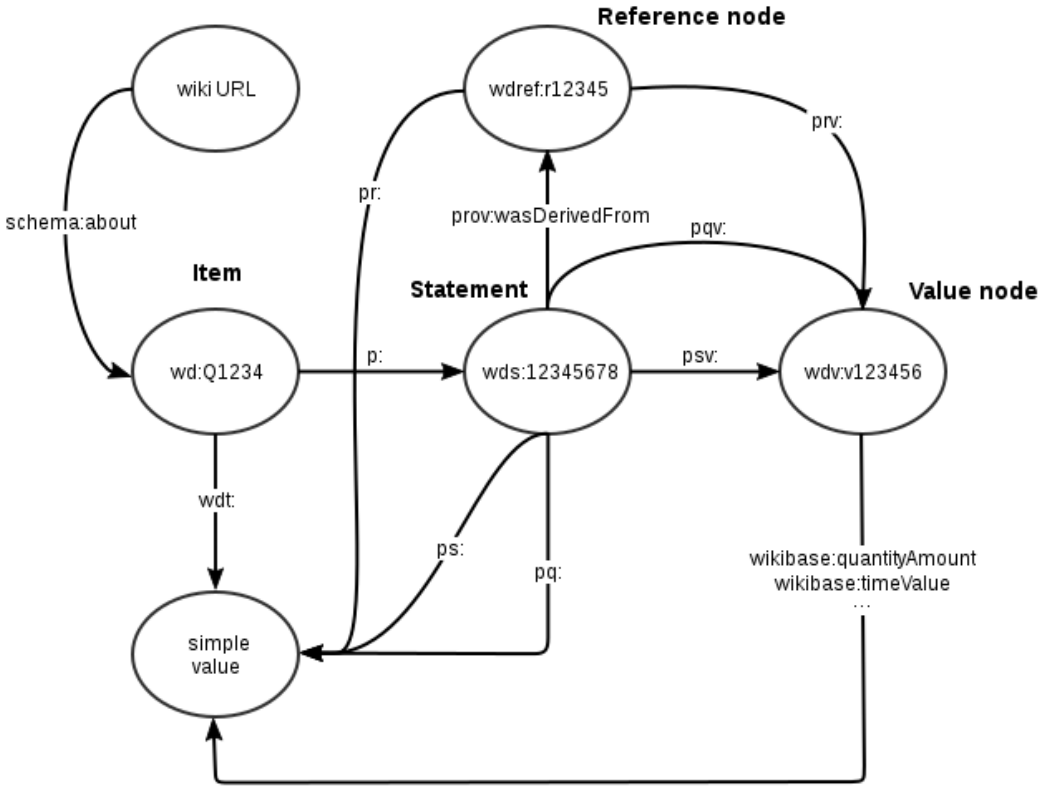

Fig. 1. Wikibase RDF model (from https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format). See Table 1 for prefix definitions.

[8] or explored via the Wikidata Query Service [3] SPARQL endpoint[4]. However, because the topology of the graph model is complex, it is not easy for humans to store and review subgraphs of the Wikidata knowledge graph acquired by these methods without significant technical expertise or access to specialized computing infrastructure such as a triplestore. Snapshots of parts of the Wikidata graph could be stored in a local Wikibase [9] installation, but that requires substantial knowledge and effort to set up the instance and to map the properties used on Wikidata with those in the local installation.

In contrast, forms of tabular data ("spreadsheets") are widely understood, easily viewed and edited using free and well-known software, and easily modified by popular scripting languages such as Python and R. Tabular data files are also easily stored and versioned by widely-used data management systems.

This paper describes a method for mapping specific Wikidata properties and the Wikibase model generally to flat comma-separated values (CSV) files using a W3C Recommendation (i.e. standard), making it possible for humans to interact more easily with the data locally. The method was developed as part of an ongoing project of the Vanderbilt University Jean and Alexander Heard Libraries known as VanderBot[5]. Although other code libraries such as Pywikibot [6] and WikidataIntegrator[7] make it possible to interact with the Wikidata API and Query Service programmatically, the simplicity of storing and manipulating data in spreadsheets and describing those data using an international standard motivated the development of this different approach.

The remainder of the paper is organized as follows. Section 2 describes how the Recommendation is applied to the Wikibase model. Section 3 explains how the method can be used to manage data in Wikidata that is of interest to a community. Section 4 describes how the method can satisfy three important use cases, and Section 5 concludes by discussing circumstances under which the method is likely to be most useful.

2. Applying the *Generating RDF from Tabular Data on the Web* Recommendation to the Wikibase model

Generating RDF from Tabular Data on the Web (CSV2RDF) [10] is a W3C Recommendation that specifies how tabular data should be processed to emit RDF triples. It is part of a broader W3C specification that defines a tabular data model [11]. The Recommendation prescribes how to construct a JSON metadata description that maps the columns of a tabular data file (e.g. CSV) to RDF triples. A conformant processor operating in "minimal mode" will convert data from the cells of the table into RDF triples based on the metadata description mappings.

Since the general Wikibase model can be expressed as RDF, the CSV2RDF Recommendation makes it possible to unambiguously describe how the contents of a simple CSV file are related to statements, qualifiers, and references in Wikidata. Thus, a CSV data file coupled with a JSON metadata description file can represent a snapshot of some subgraph of the Wikidata knowledge graph at a particular moment in time. If those snapshots were managed using version control, it would be possible to record the history of some subgraph of Wikidata as it develops over time.

Because data in Wikidata form a graph with no limit to the number of edges arising from a node, they cannot reasonably be represented as a single flat table without placing restrictions on the properties included, the number of allowed values, and the number and properties of references and qualifiers associated with the statements. Nevertheless, communities that want to monitor the state of certain types of items will generally be able to restrict the scope of the statements, references, and qualifiers to a small number that they are interested in tracking.

Mapping simple features of the Wikibase model. CSV2RDF metadata description JSON describes each column of the table by specifying its place in an RDF triple. In the following example, the subject QID (or "Q identifier") [12] for a Wikidata item is in one column of the CSV table and the value of the English label associated with that item is in another column:

```
Q_ID,English_label
Q98569123,Arthur S. Rosenberger
Q98569121,Samuel K. Mosiman
```

[3] https://query.wikidata.org

[4] https://query.wikidata.org/sparql

[5] https://github.com/HeardLibrary/linked-data/tree/master/vanderbot

[6] https://pypi.org/project/pywikibot/

[7] https://github.com/SuLab/WikidataIntegrator

The column description part of the metadata description describes how these two columns are used to generate triples. (See Appendix A for examples of the full metadata description JSON.)

```
{
"columns": [
  {
    "titles":"Q_ID",
    "name":"qid",
    "datatype":"string",
    "suppressOutput":true
  },
  {
    "titles":"English_label",
    "name":"labelEn",
    "datatype":"string",
"aboutUrl":"http://www.wikidata.org/entity
/{qid}",
    "propertyUrl":"rdfs:label",
    "lang":"en"
  }
]
}
```

The description of the first column, whose header is `Q_ID`, does not directly generate a triple since output is suppressed for it. Its description defines the variable `qid`. The value of the column denoted by that variable can be substituted into a URI template [13] as in the case of the `aboutUrl` value for the second column. The description of the second column, which contains the English label for the item, generates a triple whose subject is specified by the `aboutUrl` URI template, whose predicate has the fixed value of the `propertyUrl` JSON object, and whose literal object is the value of the `labelEn` column itself. When present, the value of the `lang` JSON object is used as the language tag for the literal value.

If the CSV table were processed using the metadata description JSON, the following graph (serialized as RDF/Turtle) would result:

```
wd:Q98569123 rdfs:label "Arthur S. Rosenberger"@en.
wd:Q98569121 rdfs:label "Samuel K. Mosiman"@en.
```

Wikidata descriptions are handled similarly to labels except that the property `schema:description` replaces `rdfs:label`.

Example 1 in Appendix A illustrates how RDF for a Wikidata statement can be generated using a CSV2RDF metadata description. In the CSV table, the second column contains the UUID identifier assigned to the statement instance and the third column contains the QID for the item that is the simple value of the statement (where `Q5` is the item for "human"). The column description maps those three columns to the Wikibase model. The objects of the triples emitted from the second and third columns are not literals, so the second and third column descriptions include an additional JSON object named `valueUrl` whose value is an IRI generated using a URI template. Note that variables used in the template *may* contain variables defined in other columns but *must* contain a variable denoting the value of the column being described. For example, the IRI identifying the statement instance is generated by combining the QID of the subject item with the UUID of the statement.

Processing the CSV file using these column descriptions emits four triples: one for each of the two data columns in each of the two rows. The predicates `p:P31` and `ps:P31` share the same local name `P31` because they represent the same property ("instance of") linking to and from the statement node.

Qualifiers are a feature of the Wikibase model that provide additional context to a statement, such as indicating the time over which the statement was valid. Since qualifier properties link statement instances to values, CSV columns containing qualifier values will be described similarly to the third column in Example 1, with the difference being that the qualifier properties will have the namespace `pq:` rather than `ps:`.

One deficiency of the CSV2RDF Recommendation is that it does not allow a column of a table to be used to generate the object of two triples. Because of that restriction, it is not possible to directly generate all possible edges defined in the Wikibase model. In Example 1, it is understood that the truthy statement triple that directly links the subject item `wd:Q98569123` to the simple value `wd:Q5` by the property `wdt:P31` exists (Fig. 1). But since the "reified" path through the statement node is already using the value of the third column (`Q5`) to generate the triple

```
wds:Q98569123-EFF1EFC4-7ECD-48E0-BE78-
CB3DB55136B1 ps:P31 wd:Q5.
```

the third column value can't also be used to generate the truthy statement triple. Fortunately, shorter paths that are understood to exist based on the Wikidata model can be constructed from longer paths using SPARQL Query [14] `CONSTRUCT` as described in Section 3.

Mapping references. A statement may have zero to many references that are used to indicate the provenance of the statement. In the Wikibase RDF export

format, statement nodes are linked to reference nodes by the property `prov:wasDerivedFrom`. Reference nodes are described by property/value pairs similar to those that describe a statement, with a difference being that there can be one to many property/value pairs describing a reference, while a statement can only have one pair.

The IRI identifier for the reference node is formed from a hash generated from the data describing the reference. Thus, any references having an identical set of property/value pairs will share the same IRI. Reference IRIs are not unique to a particular statement in the way that statements are unique to a particular item -- the same reference may serve as a source of many statements.

Example 2 in Appendix A shows how triples describing a reference for a statement would be generated from CSV tabular data.

The emitted graph consists of only three distinct triples even though there are four data cells in the CSV table. Because both statements are derived from the same source, they share the same reference instance and only a single triple describing the reference is emitted.

Mapping complex values. Some values, such as times, quantities, and globecoordinates, cannot be represented in the Wikibase model by a single literal or IRI. In those cases, an additional node, known as a *value node* (Fig. 1), must be established to link the set of triples required to fully describe the value. These value nodes are assigned a hash to uniquely identify them. In a manner similar to reference hash identifiers, complex values that are identical share the same hashes. However, there is no way to acquire the value of the hash using the Wikidata API, so for the purposes of using a CSV2RDF column description to generate RDF, we treat those nodes as blank nodes identified with Skolem IRIs [15]. To make the Skolem IRIs globally unique, a UUID may be generated as the value in the CSV column for the complex value node identifier.

Unlike general properties that are defined independently in each Wikibase instance (e.g. `P31` "instance of" in Wikidata may have a different meaning in a different Wikibase), the properties that describe complex value types are defined as part of the Wikibase model itself by the Wikibase ontology[8]. The specific properties for each complex value type will depend on whether it is a time, quantity, or globecoordinate.

Complex value nodes may be the objects of triples describing statements, references, or qualifiers (Fig. 1). The only difference is the namespace of the property serving as the predicate in the triple.

Example 3 in Appendix A illustrates a complex date value serving as the value for a "start time" (`P580`) qualifier of a "position held" (`P39`) statement.

The output graph for the example shows that the path from the statement to the time value traverses two edges (statement node to anonymous value node to time value literal). The Wikibase model implies that for qualifiers there is a direct link from the statement node to a simple value via a triple having a predicate in the `pq:` namespace (Fig. 1). The simple value depends on the kind of complex value -- for times, the simple value is the dateTime literal used in the complex value. As with truthy statements, since the CSV2RDF Recommendation does not allow one column to be used to emit two triples, both paths from the statement node to the simple value literal cannot be generated using the data in the `positionHeld_startTime_val` column. However, a SPARQL `CONSTRUCT` or `INSERT` query can be used to generate the shorter path from the longer one as demonstrated in Example 4.

Mapping language-tagged literal values. Wikibase defines a "monolingual text" value type whose value is a string literal with an associated language. Monolingual text values are represented in RDF as language-tagged literals. They can therefore be mapped using a column description similar to the one used in the label example, but with `rdfs:label` being replaced with an appropriate Wikidata property. One deficiency of the CSV2RDF Recommendation is that it does not allow the value of a `lang` JSON object in a metadata description to be a variable (variables can only be used in URI templates). The implication of the requirement to hard-code the language in the column description is that there must be a separate column in the CSV table for every language for which the user wishes to provide values.

Other features of the Wikibase model. There are several other features of the Wikibase model that have not been discussed here, such as ranks, Somevalue, Novalue, sitelinks, lexemes, and normalized values. Since these features are all included in the RDF representation of the Wikibase model, in principle they could be mapped by CSV2RDF metadata description JSON. However, since they are less commonly used

[8] http://wikiba.se/ontology

than the features described above, they are not discussed in this paper.

3. Using CSV tables to manage data in Wikidata

Using the CSV2RDF Recommendation to map CSV tables to the Wikibase model makes it possible for a community to implement a relatively simple system to write, document, version, and monitor changes to parts of the Wikidata knowledge graph.

Writing data. Prior to writing data to Wikidata, a community must decide what properties, qualifiers, and references they wish to provide and monitor for items they will track. A GUI tool[9] has been created for generating the necessary JSON metadata description and corresponding CSV column headers after selecting appropriate properties, qualifiers, and references.

If all of the items of interest are known to not exist in Wikidata, then the columns in the CSV can simply be filled in using an appropriate data source. However, more commonly, some of the items may already exist. In that case a disambiguation step should be performed to ensure that duplicate item records are not created.

Once existing items are identified, the Wikidata Query Service SPARQL endpoint can be used to download the existing statements and references for those items and save them in a CSV file whose headers correspond to the column names in the metadata description file[10]. The item QID, statement UUID, and reference hash columns in the table can be used to keep track of which data already exist in Wikidata. If data are downloaded with these identifiers, a script can know that those data do not need to be written.

After the existing data are recorded in the table, available data sources can be used to supply data for parts of the table that are missing in existing items and all of the data for new items. A script can know that these data are new because they do not yet have assigned identifiers in the table.

A Python script[11] can be used to loop through the rows of the table, using the column descriptions in the metadata description file to construct JSON in the form required to write to the Wikidata API [16]. As data are written, the identifier corresponding to those data is recorded in the table so that if a write error or server lag causes the script to terminate prematurely, there is a record of which data have been written and which data remain unwritten. At the end of the writing process, the CSV file contains data that, in combination with the metadata description file, correspond to the subgraph of Wikidata being monitored.

Documenting and versioning graphs. A CSV file and its corresponding JSON metadata description file provide a self-contained snapshot of the part of the Wikidata graph being monitored by the community. Since CSV files are relatively compact and easily compressed, it is not difficult to archive a series of files to document the state of the graph over time. For relatively small graphs, versioning can be done using readily available systems like GitHub.

At any time, the archived CSVs can be transformed by a CSV2RDF-compliant application into an RDF serialization using the metadata description file. One such application that is freely available is the Ruby application rdf-tabular [12]. The syntax for emitting RDF/Turtle from a CSV file using a metadata description file with rdf-tabular is shown in Example 1 of Appendix A.

The output file can be loaded into a triplestore using the SPARQL Update [17] `LOAD` command or can be ingested by an application that can consume RDF triples.

Generating missing triples. Because a column in a CSV cannot be used to generate more than one triple, replicating all paths in the Wikibase data model requires constructing shorter paths between nodes from longer ones. For example, truthy statement triples can be generated using SPARQL Query `CONSTRUCT` based on triple patterns that traverse the statement node (Fig. 1; namespaces declarations are omitted for brevity, see Table 1).

```
  CONSTRUCT {?item ?truthyProp ?value.}
  WHERE {
    ?item ?p ?statement.
    ?statement ?ps ?value.

FILTER(SUBSTR(STR(?ps),1,40)="http://www.wik
idata.org/prop/statement/P")
    BIND(SUBSTR(STR(?ps),40) AS ?id)

BIND(IRI(CONCAT("http://www.wikidata.org/pro
p/direct/", ?id)) AS ?truthyProp)
    }
```

[9] Code at https://github.com/HeardLibrary/linked-data/blob/master/vanderbot/wikidata-csv2rdf-metadata.html with associated .js and .css files.

[10] A Python script to download existing data for items based on a list of QIDs or screening query is available at https://github.com/HeardLibrary/linked-data/blob/master/vanderbot/acquire_wikidata_metadata.py

[11] https://github.com/HeardLibrary/linked-data/blob/master/vanderbot/vb6_upload_wikidata.py

[12] https://github.com/ruby-rdf/rdf-tabular

To insert the constructed triples directly into a graph in a triplestore, the SPARQL Update `INSERT` command can be used instead of SPARQL `CONSTUCT`. A similar query can be used to construct the triples linking statement or reference nodes to simple values based on the triple patterns that traverse a value node (Fig. 1). Example 4 of Appendix A provides Python code to insert short-path triples implied by the Wikibase graph model that are missing from graphs emitted directly from CSVs.

Comparing archived data to current Wikidata data. Because Wikidata subgraph snapshot RDF can be easily generated from a small CSV file and its corresponding JSON metadata description file, it is not necessary to maintain a triplestore containing the graph. Instead, the graph can be generated and loaded into the triplestore on the fly, used for analysis or quality control, then be deleted from the triplestore if desired.

A simple but powerful type of analysis that can be done using this workflow is to compare the state of a snapshot subgraph with the current state of Wikidata. Using a triplestore and SPARQL query interface like Apache Jena Fuseki[13] that supports federated queries, the same subquery can be made to the remote Wikidata Query Service endpoint as to a snapshot subgraph loaded into the local triplestore. The bindings of the two subqueries can then be compared using the SPARQL `MINUS` keyword. Depending on the direction of the `MINUS`, one can determine what data have been added to Wikidata since the snapshot was taken, or what data have been removed.

For example, the following query will determine labels of Bluffton University presidents that were added to Wikidata by other users after uploading data into the graph `http://bluffton` from a CSV. See Table 1 for namespace declarations, which are omitted for brevity.

```
SELECT distinct ?qid ?name
WHERE {
SERVICE <https://query.wikidata.org/sparql>
  {
  # P108 is employer
  # Q886151 is Bluffton Univ
  ?statement1 ps:P108  wd:Q886141 .
  ?qid p:P108 ?statement1.
  # P39 is position held,
  # Q61061 is chancellor
  ?statement2 ps:P39 wd:Q61061.
  ?qid p:P39 ?statement2.
  ?qid rdfs:label ?name.
  }
MINUS
  {
  GRAPH <http://bluffton> {
    ?statement1 ps:P108  wd:Q886141 .
    ?qid p:P108 ?statement1.
    ?statement2 ps:P39 wd:Q61061.
    ?qid p:P39 ?statement2.
    ?qid rdfs:label ?name.
    }
  }
}
ORDER BY ?qid
```

In this query, the item QIDs were determined by triple patterns limiting the bindings to chancellors (`Q61061`) who worked at Bluffton University (`Q886141`). An alternate approach is to limit the scope of the items by specifying their QIDs using the VALUES keyword as in this example:

```
SELECT distinct ?qid ?name
WHERE {
  GRAPH <http://bluffton>
  {
  VALUES ?qid {
    wd:Q98569123
    wd:Q98569129
    wd:Q98569121
    }
  ?qid rdfs:label ?name.
  }
MINUS
  {
SERVICE <https://query.wikidata.org/sparql>
    {
    VALUES ?qid
      {
      wd:Q98569123
      wd:Q98569129
      wd:Q98569121
      }
    ?qid rdfs:label ?name.
    }
  }
}
ORDER BY ?qid
```

In this example, the direction of the `MINUS` operation is reversed, so it would find items whose label values recorded in the local database are no longer present in Wikidata (i.e. have been changed or deleted by other users).

Some typical kinds of data to be compared are labels, values for statements involving particular properties, and reference instances.

Because the format of a federated query to a remote endpoint is very similar to the format of a query limited to a particular graph in the local triplestore, the

[13] https://jena.apache.org/documentation/fuseki2/

Table 2
Size and speed data for datasets of varying sizes

dataset	items	columns	CSV file size in kB (uncompressed / zip compressed)	conversion time (s)	triples	Turtle file size in kB (uncompressed / zip compressed)
Bluffton presidents	10	32	9 / 3	1	193	24 / 4
academic journals	431	71	160 / 58	27	13106	1500 / 320
Vanderbilt researchers	5247	30	3900 / 960	70	62634	7900 / 1300
gallery objects	4085	158	9300 / 1700	377	209597	28000 / 4400

same approach using `MINUS` can be used to compare snapshots that are loaded into different named graphs.

Performance. The system described above has been used to manage four datasets that are subgraphs of the Wikidata knowledge graph. Table 2 summarizes performance data about those datasets.

The datasets span a range of sizes that might be typical for projects taken on by a small organization. The `Bluffton presidents` dataset is a small test dataset. Items in the `academic journals` dataset include all journals in which faculty of the Vanderbilt Divinity School have published. The `Vanderbilt researchers` dataset includes nearly all current researchers and scholars affiliated with Vanderbilt University, and the `gallery objects` dataset includes the majority of items in the Vanderbilt Fine Arts Gallery. The conversion with rdf-tabulator was done under macOS 11.0.1 using a 2.3 GHz quad core processor with 8 GB memory.

Although there is some variability in the improvement in efficiency of storing the graph as a CSV over RDF/Turtle serialization, there is a clear advantage of storing the data in tabular form. In uncompressed form, raw Turtle files take approximately 3 times as much space to store. The advantage is less if the files are compressed as .zip files, with the compressed Turtle files taking roughly twice as much space.

The advantage gained in file space from storing the data as CSV is clearly outweighed by the disadvantage in time that it takes to convert the CSV files into RDF. In these trials, the conversion rate was roughly 500 triples per second. Although that rate would probably increase if the conversion were done on a more powerful computer, it is clear that the conversion time would become very limiting if graph size reached millions of triples.

Compared to the time required to convert the CSVs to Turtle, the time required to materialize the triples for the more direct paths in the Wikibase model was negligible. Those added triples increased the total number of triples in the graphs by about 35%, but even in the largest graph tested, it took only a few seconds for the SPARQL processor to construct them.

4. Applications

The ability to easily store a specific subset of Wikidata statements and references as a versioned snapshot, and to easily compare those snapshots to the current status of Wikidata makes it possible to satisfy several important use cases.

Detecting and reverting vandalism. Because anyone can edit Wikidata, the question of how to detect and revert vandalism is more important than in a knowledge graph where write access is more restricted. The system for comparing archived snapshots to the current state of Wikidata makes it possible for an organization that manages authoritative data about items (e.g. the Fine Arts Gallery dataset) to detect when those data have been changed. The system previously described would make it easy for the organization to conduct regular surveillance on statements they made about items in their collections to determine whether their authoritative data had been changed or deleted. Similarly, a university or institute might monitor statements about the educational background and creative works of researchers and scholars affiliated with them to detect unexpected changes or deletions.

Discovering value added by the community. A positive aspect of the ability of anyone to edit Wikidata is that individuals outside of an organization may improve the quality of data about items of interest to that organization. For example, authors of publications or creators of artwork may only be identified by name strings. Individuals may link those works to author or creator items, either by discovering existing items and linking to them, or by creating those items. Other examples of value added by supplementing basic data are: adding references, adding or correcting

labels in additional languages, identifying subjects of paintings or written works, and making links to other items whose relationship might not be obvious. These types of positive contributions can be discovered by the organization that initially provided the basic data using the methods outlined above. The ability to acquire these kinds of data can incentivize smaller organizations to support and contribute to Wikidata, since they may not have the financial resources to acquire those data on their own.

Future-proofing archived data. Libraries, archives, and museums have a vested interest in ensuring the sustainability of digital information. The approach described here for archiving linked open data from Wikidata meets the criteria of best practices for sustainability of digital formats [18]. As a W3C Recommendation, the CSV2RDF Recommendation fully discloses how the metadata description file should be interpreted by applications, and that file combined with the CSV makes the data self-documenting. CSV files are widely used and understood, are a preferred archival format [19], and they have a high degree of transparency since they are both easily ingested by many applications and can easily be read and understood by humans using widely available editors. Unlike the QuickStatements format[14] commonly used to upload data to Wikidata, which requires specific opaque headers for its CSV syntax, the CSV2RDF Recommendation allows column headers to be human readable labels. A future user of the archived data could have a basic understanding of the content of the data without specific knowledge of Wikidata. With the metadata description file and a copy of the CSV2RDF specification, the user could also understand clearly the relationships between the columns without detailed knowledge of the Wikibase model.

5. Conclusions

The W3C *Generating RDF from Tabular Data on the Web* Recommendation provides a mechanism for mapping simple CSV spreadsheets to the Wikibase model. That makes it possible to archive versioned snapshots of part of Wikidata in a form that is both easily examined by humans and readily accessed by commonly used software. When a CSV file is coupled with a metadata description file that maps its columns to parts of the Wikibase model, it is a faithful representation of a subgraph of the Wikidata knowledge graph at a particular time. Using a federated SPARQL query, that representation can be compared with the current state of Wikidata to detect both vandalism and value added by the community.

Because of the simplicity and small size of CSV files, this system could be particularly useful for archiving numerous snapshots of relatively small graphs. However, because of the relatively long conversion time to RDF when CSV file sizes exceed 10 MB, the system would be less useful for datasets exceeding one million triples. The advantage of the CSV format in human readability and ease of editing would also be lost once table size exceeded what can reasonably be edited using typical spreadsheet software. Nevertheless, this system could be very useful for communities that seek to manage data about Wikidata items on the order of a few thousand items and that do not have access to extensive technical infrastructure and expertise.

Acknowledgements

Gregg Kellogg provided valuable information about using the *Generating RDF from Tabular Data on the Web* Recommendation and implementation of the rdf-tabulator application.

[14] https://www.wikidata.org/wiki/Help:QuickStatements

Appendix A. Example listings
Files available at https://github.com/HeardLibrary/linked-data/tree/master/swj

Example 1. Generating RDF for a Wikidata statement using the W3C *Generating RDF from Tabular Data on the Web* Recommendation.

For a more extensive example, see https://github.com/HeardLibrary/linked-data/blob/master/json_schema/csv-metadata.json and other files in the same directory.

CSV table data (file name `bluffton_presidents.csv`):

```
Q_ID,instanceOf_uuid,instanceOf
Q98569123,EFF1EFC4-7ECD-48E0-BE78-CB3DB55136B1,Q5
Q98569121,48750747-669C-4124-BB3C-EE8F8559A265,Q5
```

Column metadata description JSON (file name `csv-metadata.json`):

```
{
"@type": "TableGroup",
"@context": "http://www.w3.org/ns/csvw",
"tables": [
  {
  "url": "bluffton_presidents.csv",
  "tableSchema": {
    "columns": [
      {
      "titles":"Q_ID",
      "name":"qid",
      "datatype":"string",
      "suppressOutput":true
      },
      {
      "titles":"instanceOf_uuid",
      "name":"instanceOf_uuid",
      "datatype":"string",
      "aboutUrl":"http://www.wikidata.org/entity/{qid}",
      "propertyUrl":"http://www.wikidata.org/prop/P31",
      "valueUrl":"http://www.wikidata.org/entity/statement/{qid}-{instanceOf_uuid}"
      },
      {
      "titles":"instanceOf",
      "name":"instanceOf",
      "datatype":"string",
      "aboutUrl":"http://www.wikidata.org/entity/statement/{qid}-{instanceOf_uuid}",
      "propertyUrl":"http://www.wikidata.org/prop/statement/P31",
      "valueUrl":"http://www.wikidata.org/entity/{instanceOf}"
      }
    ]}
  }
]}
```

`rdf-tabulator` command line to emit RDF/Turtle:

```
rdf serialize --input-format tabular --output-format ttl --metadata csv-metadata.json --minimal
```

Emitted graph (Turtle serialization, prefix declarations omitted for brevity):

```
wd:Q98569123 p:P31 wds:Q98569123-EFF1EFC4-7ECD-48E0-BE78-CB3DB55136B1.
wds:Q98569123-EFF1EFC4-7ECD-48E0-BE78-CB3DB55136B1 ps:P31 wd:Q5.
wd:Q98569121 p:P31 wds:Q98569121-EFF1EFC4-7ECD-48E0-BE78-CB3DB55136B1.
wds:Q98569121-EFF1EFC4-7ECD-48E0-BE78-CB3DB55136B1 ps:P31 wd:Q5.
```

Example 2. Generating RDF for a Wikidata reference

CSV table data (file name `bluffton_employees.csv`):

```
  Q_ID,employer_uuid,employer_ref1_hash,employer_ref1_referenceUrl
  Q98569123,D6C927AD-64B1-4212-A3EA-7FB6309F5A96,8201f36c07b460d76ca1b61a2cca6d09913500fd,
https://www.bluffton.edu/about/leadership/past-presidents.aspx
  Q98569121,1A5B3DE9-92D1-41DD-9AFF-66701CA2892B,8201f36c07b460d76ca1b61a2cca6d09913500fd,
https://www.bluffton.edu/about/leadership/past-presidents.aspx
```

Column metadata description JSON (file name `csv-metadata.json`):

```
  {
  "@type": "TableGroup",
  "@context": "http://www.w3.org/ns/csvw",
  "tables": [
    {
    "url": "bluffton_employees.csv",
    "tableSchema": {
      "columns": [
        {
        "titles":"Q_ID",
        "name":"qid",
        "datatype":"string",
        "suppressOutput":true
        } ,
        {
        "titles":"employer_uuid",
        "name":"employer_uuid",
        "datatype":"string",
        "suppressOutput":true
        },
        {
        "titles":"employer_ref1_hash",
        "name":"employer_ref1_hash",
        "datatype":"string",
        "aboutUrl":"http://www.wikidata.org/entity/statement/{qid}-{employer_uuid}",
        "propertyUrl":"prov:wasDerivedFrom",
        "valueUrl":"http://www.wikidata.org/reference/{employer_ref1_hash}"
        },
        {
        "titles":"employer_ref1_referenceUrl",
        "name":"employer_ref1_referenceUrl",
        "datatype":"string",
        "aboutUrl":"http://www.wikidata.org/reference/{employer_ref1_hash}",
        "propertyUrl":"http://www.wikidata.org/prop/reference/P854",
        "valueUrl":"{+employer_ref1_referenceUrl}"
        }
      ]}
    }
  ]}
```

Emitted graph (Turtle serialization, prefix declarations omitted for brevity):

```
  wds:Q98569123-D6C927AD-64B1-4212-A3EA-7FB6309F5A96 prov:wasDerivedFrom
wdref:8201f36c07b460d76ca1b61a2cca6d09913500fd.
  wds:Q98569121-1A5B3DE9-92D1-41DD-9AFF-66701CA2892B prov:wasDerivedFrom
wdref:8201f36c07b460d76ca1b61a2cca6d09913500fd.
  wdref:8201f36c07b460d76ca1b61a2cca6d09913500fd pr:P854
<https://www.bluffton.edu/about/leadership/past-presidents.aspx>.
```

Example 3. Generating RDF for a complex value of a Wikidata qualifier

CSV table data (file name `bluffton_positions.csv`):

```
  Q_ID,positionHeld_uuid,positionHeld_startTime_nodeId,positionHeld_startTime_val,
positionHeld_startTime_prec
  Q98569123,5B56773B-8730-4A9D-AD85-78F7ABA17225,3cf7cfe7-ee1d-49a9-b493-6a315b0ec219,
1935-01-01T00:00:00Z,9
```

Column metadata description JSON (file name `csv-metadata.json`):

```
  {
  "@type": "TableGroup",
  "@context": "http://www.w3.org/ns/csvw",
  "tables": [
    {
    "url": "bluffton_positions.csv",
    "tableSchema": {
      "columns": [
        {
        "titles":"Q_ID",
        "name":"qid",
        "datatype":"string",
        "suppressOutput":true
        },
        {
        "titles":"positionHeld_uuid",
        "name":"positionHeld_uuid",
        "datatype":"string",
        "aboutUrl":"http://www.wikidata.org/entity/{qid}",
        "propertyUrl":"http://www.wikidata.org/prop/P39",
        "valueUrl":"http://www.wikidata.org/entity/statement/{qid}-{positionHeld_uuid}"
        },
        {
       "titles":"positionHeld_startTime_nodeId",
        "name":"positionHeld_startTime_nodeId",
        "datatype":"string",
        "aboutUrl":"http://www.wikidata.org/entity/statement/{qid}-{positionHeld_uuid}",
        "propertyUrl":"http://www.wikidata.org/prop/qualifier/value/P580",
        "valueUrl":"http://example.com/.well-known/genid/{positionHeld_startTime_nodeId}"
        },
        {
        "titles":"positionHeld_startTime_val",
        "name":"positionHeld_startTime_val",
        "datatype":"dateTime",
        "aboutUrl":"http://example.com/.well-known/genid/{positionHeld_startTime_nodeId}",
        "propertyUrl":"http://wikiba.se/ontology#timeValue"
        },
        {
        "titles":"positionHeld_startTime_prec",
        "name":"positionHeld_startTime_prec",
        "datatype":"integer",
        "aboutUrl":"http://example.com/.well-known/genid/{positionHeld_startTime_nodeId}",
        "propertyUrl":"http://wikiba.se/ontology#timePrecision"
        }
      ]}
    }
  ]}
```

Emitted graph (Turtle serialization, prefix declarations omitted for brevity):

```
  wd:Q98569123 p:P39 wds:Q98569123-5B56773B-8730-4A9D-AD85-78F7ABA17225.
  wds:Q98569123-5B56773B-8730-4A9D-AD85-78F7ABA17225 pqv:P580
<http://example.com/.well-known/genid/3cf7cfe7-ee1d-49a9-b493-6a315b0ec219>.
  <http://example.com/.well-known/genid/3cf7cfe7-ee1d-49a9-b493-6a315b0ec219>
wikibase:timePrecision 9;
    wikibase:timeValue "1935-01-01T00:00:00Z"^^xsd:dateTime .
```

Example 4. Python script to materialize triples for shorter alternate Wikibase paths

```
# (c) 2020 Vanderbilt University.  Author: Steve Baskauf (2020-11-28)
# This program is released under a GNU General Public License v3.0 http://www.gnu.org/licenses/gpl-3.0

import requests

# port 3030 is used by a local installation of Apache Jena Fuseki
dataset_name = 'data'
graph_iri = 'http://bluffton'
endpoint = 'http://localhost:3030/' + dataset_name + '/update'

namespaces = '''
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix prov: <http://www.w3.org/ns/prov#>
prefix wikibase: <http://wikiba.se/ontology#>
prefix  wd:  <http://www.wikidata.org/entity/>
prefix  wdt: <http://www.wikidata.org/prop/direct/>
prefix  p:  <http://www.wikidata.org/prop/>
prefix  pq:  <http://www.wikidata.org/prop/qualifier/>
prefix  pr:  <http://www.wikidata.org/prop/reference/>
prefix  ps:  <http://www.wikidata.org/prop/statement/>
prefix  pqv:  <http://www.wikidata.org/prop/qualifier/value/>
prefix  prv:  <http://www.wikidata.org/prop/reference/value/>
prefix  psv:  <http://www.wikidata.org/prop/statement/value/>
'''

value_types = [
    {'string': 'time',
     'local_names': ['timeValue'],
     'datatype':'http://www.w3.org/2001/XMLSchema#dateTime',
     'bind': '?literal0'},
    {'string': 'quantity',
     'local_names': ['quantityAmount'],
     'datatype': 'http://www.w3.org/2001/XMLSchema#decimal',
     'bind': '?literal0'},
    {'string': 'globecoordinate',
     'local_names': ['geoLatitude', 'geoLongitude'],
     'datatype': 'http://www.opengis.net/ont/geosparql#wktLiteral',
     'bind': 'concat("Point(", str(?literal0), " ", str(?literal1), ")")'}
]

property_types = ['statement', 'qualifier', 'reference']

# Insert the missing value statements using values from value nodes
for value_type in value_types:
    for property_type in property_types:
        query = '''
        WITH <''' + graph_iri + '''>
        INSERT {?reference ?directProp ?literal.}
        WHERE {
          ?reference ?pxv ?value.
        '''
        for ln_index in range(len(value_type['local_names'])):
            query += '  ?value wikibase:' + value_type['local_names'][ln_index] + ' ?literal' + str(ln_index) + '''.
        '''
        query += '  bind(' + value_type['bind'] + ''' as ?literal)
          FILTER(SUBSTR(STR(?pxv),1,45)="http://www.wikidata.org/prop/''' + property_type + '''/value/")
          BIND(SUBSTR(STR(?pxv),46) AS ?id)
          BIND(IRI(CONCAT("http://www.wikidata.org/prop/''' + property_type + '''/", ?id)) AS ?directProp)
          }
          '''
        #print(query)
        print('updating', property_type, value_type['string'])
        response = requests.post(endpoint, headers={'Content-Type': 'application/sparql-update'}, data = namespaces + query)
    print('update complete')

# Insert the missing "truthy" statements from statement value statements
query = '''
WITH <''' + graph_iri + '''>
INSERT {?item ?truthyProp ?value.}
WHERE {
  ?item ?p ?statement.
  ?statement ?ps ?value.
  FILTER(SUBSTR(STR(?ps),1,40)="http://www.wikidata.org/prop/statement/P")
  BIND(SUBSTR(STR(?ps),40) AS ?id)
  BIND(IRI(CONCAT("http://www.wikidata.org/prop/direct/", ?id)) AS ?truthyProp)
  }
  '''
#print(query)
print ('updating truthy statements')
response = requests.post(endpoint, headers={'Content-Type': 'application/sparql-update'}, data = namespaces + query)
print('done')
```

References

[1] M. Mora-Cantallops, S. Sánchez-Alonso and E. García-Barriocanal, A systematic literature review on Wikidata, Data Technologies and Applications 53 (3) 250-268, 2019. doi:10.1108/DTA-12-2018-0110

[2] Wikimedia Foundation, Wikidata:Bots, https://www.wikidata.org/wiki/Wikidata:Bots [Accessed 22 December 2020]

[3] Wikimedia Foundation, GLAM, Wikimedia Meta-Wiki, https://meta.wikimedia.org/wiki/GLAM [Accessed 22 December 2020]

[4] Meetup/Cost MOBILISE Wikidata Workshop, 2020. https://en.wikipedia.org/wiki/Wikipedia:Meetup/Cost_MOBILISE_Wikidata_Workshop [Accessed 22 December 2020]

[5] LD4-Wikidata Affinity Group, https://wiki.lyrasis.org/display/LD4P2/LD4-Wikidata+Affinity+Group [Accessed 22 December 2020]

[6] Wikimedia Foundation, Wikibase/DataModel, https://www.mediawiki.org/wiki/Wikibase/DataModel [Accessed 22 December 2020]

[7] S. Malyshev and G. Lederrey, Wikibase/Indexing/RDF Dump Format, https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format [Accessed 22 December 2020]

[8] Wikimedia Foundation, Wikidata:Data access, https://www.wikidata.org/wiki/Wikidata:Data_access#Per-item_access_to_data. [Accessed 22 December 2020]

[9] Wikimedia Germany, Welcome to the Wikibase project, https://wikiba.se/ [Accessed 22 December 2020]

[10] J. Tandy, I. Herman and G. Kellogg, Generating RDF from Tabular Data on the Web W3C Recommendation, 17 December 2015. http://www.w3.org/TR/csv2rdf/

[11] J. Tennison, G. Kellogg and I. Herman, Model for Tabular Data and Metadata on the Web W3C Recommendation, 17 December 2015. http://www.w3.org/TR/tabular-data-model/

[12] Wikimedia Foundation, Wikidata:Glossary, https://www.wikidata.org/wiki/Wikidata:Glossary [Accessed 22 December 2020]

[13] J. Gregorio, R. Fielding, M. Hadley, M. Nottingham and D. Ochard, URI Template, March 2012. https://tools.ietf.org/html/rfc6570

[14] S. Harris, A. Seaborne and E. Prud'hommeaux, SPARQL 1.1 Query Language, 21 March 2013. https://www.w3.org/TR/sparql11-query/

[15] W3C, RDF 1.1 Concepts and Abstract Syntax, 25 February 2014. https://www.w3.org/TR/rdf11-concepts/#section-skolemization

[16] Wikimedia Foundation, Wikibase/DataModel/JSON, https://www.mediawiki.org/wiki/Wikibase/DataModel/JSON [Accessed 22 December 2020]

[17] P. Gearon, A. Passant and A. Polleres, SPARQL 1.1 Update W3C Recommendation, 21 March 2013. https://www.w3.org/TR/sparql11-update/

[18] Library of Congress, Sustainability of Digital Formats: Planning for Library of Congress Collections, 5 January 2017. https://www.loc.gov/preservation/digital/formats/sustain/sustain.shtml [Accessed 20 December 2020]

[19] Library of Congress, Recommended Formats Statement. VI. Datasets, https://www.loc.gov/preservation/resources/rfs/data.html#datasets [Accessed 21 December 2020]