Semantic Web 14 (2023) 5–27
DOI 10.3233/SW-210443
IOS Press

Using the W3C *Generating RDF from Tabular Data on the Web* recommendation to manage small Wikidata datasets

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; ORCID: https://orcid.org/0000-0003-4365-3135
[b] *Carleton College, Northfield, Minnesota, USA*
ORCID: https://orcid.org/0000-0002-1772-1045

Editors: Jose Emilio Labra Gayo, University of Oviedo, Spain; Anastasia Dimou, IDLab, Ghent University, Belgium; Katherine Thornton, Yale University Library, USA; Anisa Rula, University of Milano-Bicocca, Italy and University of Bonn, Germany
Solicited reviews: Jakub Klimek, Charles University, Czech Republic; John Samuel, CPE Lyon, France; Andra Waagmeester, Maastricht University, Netherlands; Tom Baker, Sungkyunkwan University, South Korea; Dimitris Kontokostas, Universität Leipzig, Germany

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 used by Wikidata 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 [21]. Its dedicated users and easy-to-use graphical interface promise value added through community contributions, and access through its API[1] makes it possible for large amounts of data to be contributed via bot scripts [29] that upload data from a variety of sources.

These advantages have generated a lot of interest among biodiversity informaticians [20], information specialists [15], and others in communities such as galleries, libraries, archives and museums (GLAM) [30], for using Wikidata as a place to expose and manage data about items of their concern, such as collections records, authors, and authority

*Corresponding author. E-mail: steve.baskauf@vanderbilt.edu.
**Now at Google.
[1] https://www.wikidata.org/w/api.php

files. Because those data are exposed as RDF through the Wikidata Query Service,[2] Wikidata provides an entrance to the Linked Open Data (LOD) cloud[3] with a relatively low barrier to entry.

In this paper, we argue that Wikidata provides an opportunity for user groups and small institutions with limited technical expertise to participate in the LOD movement if simple systems are available for them to upload and monitor data about items in which they have a vested interest (referred to henceforth as "items of interest"). We also argue that for users at small GLAM institutions with limited IT resources, a system based on comma-separated tabular data ("CSV") files is not only simple but is the most appropriate format given that community's long-term interest in archival preservation of data. Although there are existing systems for writing tabular data to Wikidata, they do not fully document the semantics of the table columns in a manner that allows for easy reconstruction of RDF that fully describes the subgraph of Wikidata serialized in the table.

We describe a method for mapping specific Wikidata properties and the Wikibase model generally to flat CSV files using the *Generating RDF from Tabular Data on the Web* (CSV2RDF) W3C Recommendation (i.e. standard), making it possible for humans to interact more easily with the data locally using spreadsheet software. The method was developed as part of an ongoing project of the Vanderbilt University Jean and Alexander Heard Libraries known as *VanderBot*,[4] whose goal was to allow human Wikidata editors to submit data more efficiently than through using the graphical interface. Although other code libraries such as *Pywikibot*[5] and *WikidataIntegrator*[6] make it possible to interact with the Wikidata API and Query Service programmatically, they require significant experience with coding and software configuration. In contrast, the *VanderBot* Python scripts based on the model presented here require no coding experience and minimal installation effort.

The main novelty of this work is demonstrating a single system that simultaneously:

- is extremely simple and easily used by those with a limited computer science background
- maintains the ability to fully capture details of a subgraph of Wikidata in a form that is easily archived
- can easily be ingested or output by scripts
- can be used to track subgraph changes over time.

The simplicity of the system does come at some costs imposed by the limitations of the CSV2RDF specification itself, including the inability to express the language of literals as part of the data or to generate two triples from the same column of data (necessitating some post-processing). General limitations of "flat" tables also apply to this approach, such as complications when a property has multiple values for a single item. However, we believe that these shortcomings are outweighed by the benefits gained from the wide acceptance of CSV as an archival format and from allowing non-technical users to edit, copy, and paste data using familiar spreadsheet software, as well as scan columns for patterns and missing data.

This paper is organized as follows. The remainder of this introduction describes the key features of the Wikibase model and its Wikidata implementation. Section 2 reviews related work. Section 3 describes how the CSV2RDF Recommendation is applied to the Wikibase model. Section 4 explains how the method can be used to manage Wikidata items of interest. Section 5 describes how the method can satisfy three important use cases, and Section 6 concludes by discussing circumstances under which the method is likely to be most useful.

Features of Wikidata. Wikidata has several features that distinguish it from more generic LOD: a specified graph model, a simplified approach to RDF, and a centralized repository.

Wikidata is built on a general model known as the Wikibase data model [31]. The Wikibase data model has an RDF export format [19] 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,

[2] https://query.wikidata.org
[3] https://www.lod-cloud.net/
[4] https://github.com/HeardLibrary/linked-data/tree/master/vanderbot
[5] https://pypi.org/project/pywikibot/
[6] https://github.com/SuLab/WikidataIntegrator

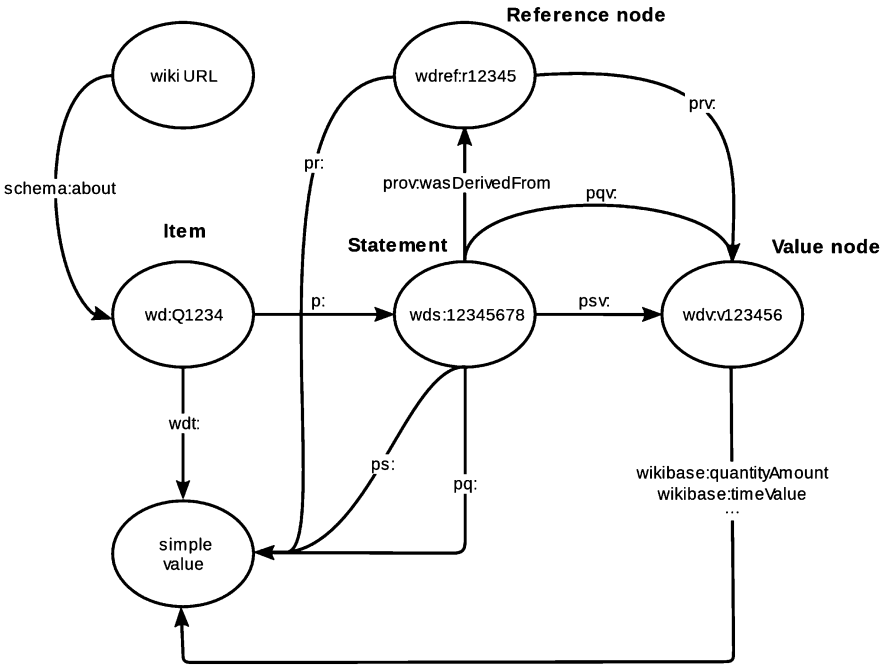


Fig. 1. Wikibase RDF model (from https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format). See Table 1 for prefix definitions. Image by Michael F. Schönitzer CC0.

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#

and *value nodes* that make it possible to describe complex values like time, quantities, and globecoordinates. 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.

Although the data in Wikidata are available in the form of RDF via a SPARQL endpoint, they are actually managed in a relational database, with periodic exports by a Query Service updater to the triplestore (i.e. specialized RDF graph database) behind the endpoint (Fig. 2, right side). However, because the JSON that is used to interact with the database via the API is structured based on the same abstract data model as the triplestore (i.e. the Wikibase data model), a client that can use that model to interpret data acquired from the triplestore through SPARQL queries

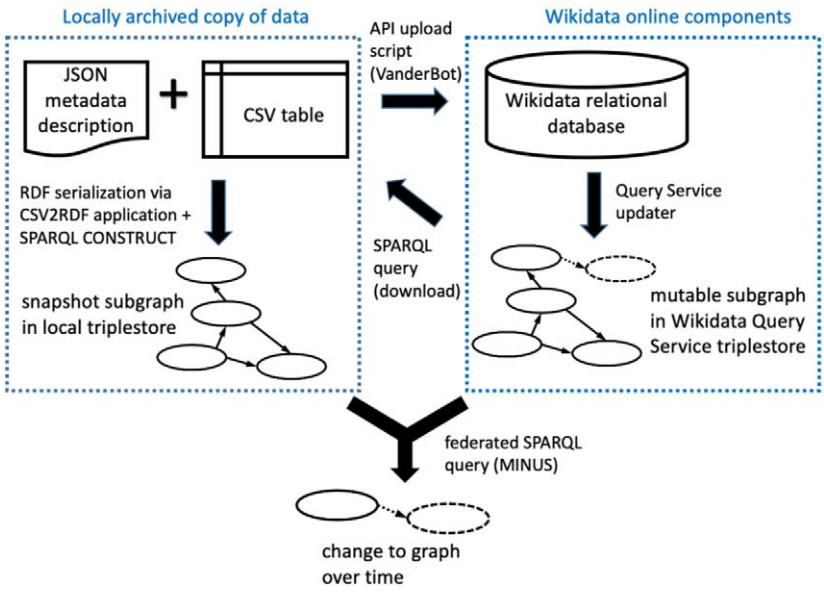


Fig. 2. Key components and interactions described in this paper. The Wikimedia Foundation hosts the online components with data maintained by the user community. Users may keep a local set of data in order to upload new data to Wikidata via the API, or as downloaded archival subgraph snapshots generated using the Query Service. A federated SPARQL query can be used to compare online and local subgraphs.

can also be designed to construct and parse the JSON necessary for API uploads involving similarly structured data.

The design of the Wikibase RDF model makes it simpler in some ways than generic LOD. The Wikibase model does not include the notion of classes as distinct entities as introduced in the RDF Schema (RDFS) model [8]. Instead, in Wikidata items that are values of the property P31 ("instance of") fill the role of classes. Thus, the class proxies are simply part of the data rather than being defined as part of an independent ontology. Similarly, properties are also part of the community-curated data rather than being defined in a separate ontology, so terminological changes over time could be documented simultaneously with changes in assertions without the need to monitor an external ontology.

Wikidata deals with the issue of tracking sources by including the notion of a statement instance (Fig. 1) as part of its data model rather than depending on special mechanisms such as reification or named graphs to generate nodes to which provenance data can be attached. Again, this simplifies the system since properties and values of qualifiers and references that are linked to statement instances are managed using the same system as properties and items used to make statements.

Wikidata also differs from traditional LOD in that it is centrally managed rather than distributed. That makes it possible to detect changes in the knowledge graph over time without needing to access multiple endpoints or dereference IRIs. Although anyone can edit items, changes to properties can only be made through community consensus, adding a degree of stability and transparency that would not be ensured if external ontologies were used.

Subgraphs of the Wikidata knowledge graph can be exported in one of the RDF serializations [32] or explored via the Wikidata Query Service SPARQL endpoint.[7] 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 [35] installation, but that requires substantial knowledge and effort to set up the instance and to map the properties used in Wikidata with those in the local installation.

In contrast, our system of storing a subgraph as a CSV file coupled with a JSON metadata description file makes it easy to review data and store it by traditional means. The combination of CSV + metadata description effectively

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

serves as an RDF serialization since it is interconvertible with other more typical serializations that could be loaded into a triplestore if desired (Fig. 2, left side).

2. Related work

Small institutions like museums, galleries, and libraries and user groups focused on relatively narrow topics are an important part of the Wikidata contributor community.[8] In this section, we make the case that success in community-driven efforts can be achieved by simpler solutions rather than more technologically complex approaches. We describe successful existing tools for editing tabular data and submitting those data to Wikidata. We review previous work related to developing best practices for archiving Linked Data and note how those recommendations might apply in the circumstances of small GLAM institutions that have created and are monitoring items of interest in Wikidata. Finally, we review several systems for versioning RDF and approaches for monitoring for vandalism in Wikidata, then discuss why these systems generally would not be effective for use by small organizations and institutions whose items of interest are part of a relatively small subgraph of the Wikidata knowledge graph.

Ease of use and its relationship to community participation. It would be tempting to assume that progress in developing a knowledge base might be achieved only through an increasing degree of technological sophistication. However, evidence has shown that in circumstances where wide community involvement is desirable, progress is more likely when technology becomes simpler rather than more complex.

The case of the technical architecture of Biodiversity Information Standards (TDWG)[9] is instructive. In 2006, the TDWG Technical Architecture Group (TAG) proposed implementation of a complex "three legged stool" architecture based on Life Science Identifiers (LSID) unique identifiers [22], OWL ontologies, and an XML-based data-exchange standard using those ontologies [27]. However, LSID technology was difficult to implement, the OWL ontologies were unmaintainable [14], and the OWL/XML transfer system was too cumbersome to be efficient and too complex to be implemented by small data providers [7]. Within five years, the "three legged stool" model was effectively dead, with virtually no use of LSIDs, effective deprecation of the TDWG Ontology [6], and the XML transfer system not widely used. Currently 97% of the over 1.7×10^9 occurrence records in the Global Biodiversity Information Facility have been published using Darwin Core Archives,[10] a simpler system based on tabular data files.[11] This includes many records provided by small natural history museums that manage their records using Excel spreadsheets and whose participation is enabled by the simplicity of the system.

Within the library community, there is wide recognition that deploying generic LOD applications is a complex task restricted primarily "to large, well-resourced institutions, often with external financial support" [3]. However, there is significant interest within the GLAM community for exploring Wikidata, evidenced by the large number of Wikidata WikiProjects in the GLAM category.[12] Available software and user-friendly Wikidata editing tools "put experimentation and implementation of linked open data within the reach of more libraries" [3].

Existing table-based tools for editing Wikidata. The popularity and widespread use of Wikidata can to a large extent be attributed to the availability of simple editing tools that can be used by novices. In addition to the easy-to-use graphical interface, users can also contribute using two relatively user-friendly tools that enable manipulation and submission of tabular data: *QuickStatements*[13] and *OpenRefine*.[14]

QuickStatements accepts plain text files as input using either an idiosyncratic "command sequence" syntax or CSV files having a particular structure. Interpretation of the CSV syntax depends on a header row that must conform to a very specific set of alphanumeric codes applied in a specified sequence. Because its input format is well-defined, *QuickStatements* input files can be created as the output of other scripts. This allows developers to create tools such

[8] https://www.wikidata.org/wiki/Wikidata:WikiProjects
[9] https://www.tdwg.org/
[10] https://ipt.gbif.org/manual/en/ipt/2.5/dwca-guide
[11] Based on a query of occurrence records performed on 2021-06-01.
[12] https://www.wikidata.org/wiki/Category:GLAM_WikiProjects
[13] https://www.wikidata.org/wiki/Help:QuickStatements
[14] https://openrefine.org/

as *Author Disambiguator*[15] that pass their output to *QuickStatements* and thus avoid the need to code the interactions with the Wikidata API.

The ability to use *QuickStatements* with CSV files provides the benefits of easy review and editing with a spreadsheet program. However, the restrictions placed on column headers do not permit the use of easy-to-understand column names. A *QuickStatements* CSV file also cannot be converted directly to a well-known RDF serialization without an application designed specifically to interpret the coded column headers. These characteristics make *QuickStatements* CSV files difficult to use for archival purposes if it is considered important to be able to reconstruct the RDF subgraph that the CSV represents.

OpenRefine is a free, well-known tool for cleaning tabular data. Its Wikidata extension[16] allows a user to transform tables (including imported CSV files) into Wikidata statements. A key feature is its Reconciliation service that allows the software to assist the user in matching their data to existing items in Wikidata. Mapping tabular data in *OpenRefine* to Wikidata statements is done using a schema that can be created or imported by the user. *OpenRefine* includes many cleaning tools and a scripting language, features which make it a powerful tool but which also increase the learning curve for using it.

Tables as edited within *OpenRefine* can be more complex than simple CSVs, with multiple rows being considered as part of a single record. CSV is available as a lossy export format, but as with *QuickStatements*, if these CSVs were archived they could not directly be translated into a common RDF serialization.

In the case of both *QuickStatements* and *OpenRefine*, the format of their associated CSV files does not have a mechanism for preserving the statement and reference node identifiers necessary to fully document the subgraphs they represent.

A less well-known tool, *Wikibase_Universal_Bot*,[17] is built on *WikidataIntegrator* but does not require the user to code. It can use CSV data as its input but requires the user to construct a configuration file and a data model in YAML format to interpret the CSV. Using it requires some understanding of WikidataIntegrator and the ability to install and configure the tool.

Preservation of Linked Open Data. Archival preservation of LOD presents special challenges that are not present in more traditional datasets. In this subsection, we will review some of the key assessments of these challenges and will show how some of the problems involved in preservation of LOD do not apply in the situation of Wikidata, making preservation of its data more tractable. We also argue that a system based on CSV files such as the one presented here makes the task even easier and is therefore a desirable approach.

Archival preservation of data is a major concern of the GLAM community. Best practices for preservation of digital data have a long history in the archival preservation community. ISO 14721 describes a reference model for an open archival information system (OAIS) for digital data [23]. This standard describes many aspects of data preservation, but the part of the model most relevant to this paper is the Information Package definition (Section 2.2.2). It differentiates between the Content Information (the information to be preserved) and the Preservation Description Information (PDI). The PDI includes a description of how the Content Information relates to information outside the Information Package, i.e. the context that makes the Content Information understandable to the user community. Section 3.2.4 describes the requirement that the Information Package must be understandable not only to the original data producers, but also to the Designated Community, which must be able to understand the information over the Long Term without the help of the experts who generated the information.

Since the development of the original OAIS model in 2003, data preservation specialists in the GLAM community have sought to understand how the model might apply in the context of LOD. Bartoli et al. [5] provided an analysis of strategies for ensuring long-term preservation of LOD. Although much of their paper focused on how standardized vocabularies can be used to describe LOD datasets, their review of the state of the art identified a key point related to preservation of the data itself: the set of tools needed to understand a dataset are different from those needed to use and query it. That is, software necessary for installing a triplestore and querying it may be more complex and difficult to preserve than the RDF that constitutes the dataset. Thus an archiving system that makes the data directly available in a form that does not require complex software for its interpretation has an advantage from a

[15] https://author-disambiguator.toolforge.org/
[16] https://www.wikidata.org/wiki/Wikidata:Tools/OpenRefine
[17] https://github.com/dcodings/Wikibase_Universal_Bot

preservation perspective. The system described in this paper is such a system because archived CSV files can be examined without specialized software that would need to be archived and documented alongside the data files.

The PRELIDA project [1] examined the gaps between the Linked Data and Digital Preservation communities to describe steps towards efficient digital preservation of linked data. They made a distinction between RDF serialized and stored across the web and RDF stored in a triplestore. Differences between these "online" and "offline" data sources result in important differences in strategies to preserve them. It is easier to preserve "snapshots" of offline data sources since obtaining those versions does not depend on dereferencing IRIs that may have suffered "link rot". In their gap analysis (their Section 4), they identified several challenges for preserving Linked Data.

However, several of these challenges do not apply to Wikidata. Since the Linked Data representation of Wikidata is accessed exclusively from sources internal to the Wikidata system, the problem of dependence on web infrastructure is largely absent. Formal knowledge problems associated with dependence on external ontologies are largely missing since the semantics of Wikidata are described almost entirely by the Wikibase ontology and graph model, and by properties whose definitions are included as part of the data within Wikidata itself. Complications involving preservation of provenance metadata associated with links are reduced by the Wikibase model, which has built-in link (i.e. statement) instantiation, allowing for reference and qualifier data to be associated with links as part of the dataset itself.

The PRELIDA project also raised several issues that can be examined in the specific context of Wikidata. One identified preservation risk was failure to preserve the ontologies that provide classes and properties that express the data. That risk was also identified in Best Practice 28 for data preservation in *Data on the Web Best Practices* [18]. In Wikidata, this risk is avoided because the Wikibase model does not rely upon classes as they are defined in RDFS, and because properties are defined as part of the data.

The PRELIDA project raised the question of who is responsible for "community" data. On the surface, in the case of Wikidata the responsible party would seem to be the Wikimedia Foundation, which operates Wikidata. But because anyone can edit Wikidata, GLAM institutions can take responsibility for adding and curating data in Wikidata about items of their concern. This effectively places them in the roles of both data producer and Designated Community in the OAIS sense, and it enables them to have a role in ensuring that the data they care about remain accurate and available.

Finally, the PRELIDA project considered the question of durability of the format of the data. Since RDF is based on open standards and can be serialized as plain text, it can be considered very durable. Wikidata also maintains its own permanent, unique IRI identifiers for items and properties, reducing problems related to IRI maintenance and dereferencing.

In considering the relative durability of data formats, the Library of Congress recommended formats for datasets [16] is instructive. Preferred formats are platform-independent and character-based. They are also well-developed, widely adopted and are de facto marketplace standards. Examples include line-oriented formats like CSV and formats using well-known schemas with public validation tools. With respect to RDF serializations, JSON-LD and RDF/XML would fall in the latter category. However, although JSON and XML are well-known data transfer formats, their use as RDF serializations is not familiar to most archivists and they are not readily interpretable without software specially designed to consume them. In contrast, CSV files are widely accepted and understood in the archival community and can be easily interpreted using widely available software. For these reasons, we argue that CSV should be considered a preferred format for archiving RDF data.

Versioning and monitoring RDF datasets. Debattista et al. identified several desirable future directions in their introduction to a 2019 special issue on "Managing the Evolution and Preservation of the Data Web" [10]. They noted that "there is a lack of practical recipes for publishing and managing evolving data and knowledge" in the context of Linked Data. That issue provided two examples of systems for managing versions of evolving RDF data: *Quit Store*[18] and *OSTRICH*.[19] However, the use cases of these solutions do not align closely with those we consider in this paper, so they have limited applicability for our use cases. The focus of the *Quit Store* system, developed by Arndt et al. [2], was on dispersed resources and not centrally managed systems like Wikidata. The same is true of other versioning and storage systems for RDF data reviewed by Arndt et al. The *OSTRICH* system developed

[18] https://github.com/AKSW/QuitStore
[19] https://github.com/rdfostrich/ostrich

by Taelman et al. [25] queried archived RDF versions using a method that optimized storage space and querying efficiency through preprocessing on the ingestion side. However, they were concerned with working at web scale, so their solution was more complex than necessary for use by GLAM institutions seeking to archive and search versions of small subgraphs from Wikidata.

Vandalism has been a longstanding concern in the Wikidata community, and a number of tools have been developed to counter it.[20] Typically, these tools assist human users who "patrol" for vandalism by looking for suspicious recent edits that conform to some negative pattern or fail to conform to some pattern expected for particular types of items. For example, statements whose values violate property constraints are suspect.[21] For some types of items, conformance to expected patterns can be formally assessed using constraint violation reports based on schemas[22] constructed using Shape Expressions (ShEx) [4]. Checking for constraint violations could be a valuable tool for vandalism detection in the GLAM community, although it would require the technical ability to create a ShEx schema for the item type to be monitored if one did not already exist. Sarabadani et al. [24] built an automated, real-time vandalism detection tool that used machine learning to increase the efficiency of patrollers. It reduced the human labor involved in patrolling edits by 98%.

These tools are generally focused on detecting Wikidata-wide vandalism. However, GLAM institutions are likely to be more focused on detecting both vandalism and value added to their items of interest rather than on the scale of all of Wikidata. Thus, methods optimized to make it easy for them to track changes over time for a specified set of items are likely to be more useful to them than Wikidata-wide tools, which incur more complexity because of their broad scope.

3. Applying the *Generating RDF from Tabular Data on the Web* recommendation to the Wikibase model

Generating RDF from Tabular Data on the Web (CSV2RDF) [26] 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 [28]. The Recommendation prescribes how to construct a JSON metadata description file 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. The JSON metadata description file describes each column of the table by specifying its place in an RDF triple. In the following example, the subject QID (or "Q identifier") [33] 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
```

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 JSON metadata description files.)

[20] https://www.wikidata.org/wiki/Wikidata:WikiProject_Counter-Vandalism
[21] https://www.wikidata.org/wiki/Help:Property_constraints_portal
[22] https://www.wikidata.org/wiki/Wikidata:WikiProject_Schemas

```
{
"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 [12] 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 JSON metadata description file, 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 in the description file except that the property `schema:description` replaces `rdfs:label`.

Example 1 in Appendix A illustrates how RDF for a Wikidata statement can be generated from CSV data using a JSON metadata description file. 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 abstract relationship ("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 assumed 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 assumed to exist based on the Wikidata model can be constructed from longer paths using SPARQL Query [13] `CONSTRUCT` as described in Section 4.

One complication of this process is that not every statement instance has a corresponding truthy statement triple. Statements with the rank "deprecated", and statements with a "normal" rank but where other values for that property have a "preferred" rank do not have corresponding truthy statement triples.[23] Since many users are unaware of ranks and many bots don't consider ranks, relatively few statements have a rank other than the default "normal". However, generating truthy statement triples based on the assumption that they exist for every statement instance can introduce artificial differences between subgraphs in Wikidata and corresponding subgraphs generated from CSVs using the method described in this paper. The possibility of these differences must be considered when monitoring for changes in subgraphs over time as described in Section 4.

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 [9]. 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.[24] 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

[23] Thanks to Andra Waagmeester for pointing out this complication.

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

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 file 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 JSON metadata description files constructed according to the CSV2RDF specification. However, since they are less commonly used than the features described above, they are not discussed in this paper.

4. Using CSV tables to manage Wikidata data

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 (Fig. 2).

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[25] has been created for generating the necessary JSON metadata description file and a corresponding CSV file 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 JSON metadata description file.[26] 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 known as *VanderBot*[27] can be used to loop through the rows of the table, using the column descriptions in the metadata description file to construct different JSON in the form required to write to the Wikidata API [34]. 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 JSON metadata description file, correspond to the subgraph of Wikidata being monitored.

[25] See Appendix B for a screenshot, a link to the code, and a link to an operational web page.

[26] 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

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

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 JSON metadata description file (Fig. 2, left side). One such application that is freely available is the Ruby application *rdf-tabular*.[28] The syntax for emitting RDF Turtle from a CSV file using a JSON 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 [11] `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).

Query 1. Construct truthy statements that cannot be generated directly from the CSV data

```
CONSTRUCT {?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)
    }
```

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[29] 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 (Fig. 2, bottom).

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.

Query 2. Detect labels in Wikidata not present in a prior version snapshot, with items restricted by a graph pattern

```
SELECT DISTINCT ?qid ?name
WHERE {
SERVICE <https://query.wikidata.org/sparql>
  {
```

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

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

```
  # 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 shown in example Query 3.

Query 3. Detect labels no longer present in Wikidata, with items restricted by designating a list of item values

```
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.

Table 2
Conversion data for datasets of varying sizes

Dataset	Items (rows)	Columns	CSV file size in kB	Conversion time (s)	Triples
Bluffton presidents	10	32	9	1	193
academic journals	431	71	160	27	13106
Vanderbilt researchers	5247	30	3900	70	62634
gallery objects	4085	158	9300	377	209597

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

We noted previously that constructing truthy statement triples for every statement instance without first considering the rank of the statement can introduce artificial differences between subgraphs generated from CSVs and their corresponding subgraphs in Wikidata. This possibility must be considered for MINUS comparisons that include truthy edges in the graph patterns used to bind solutions. To avoid this problem, it is best whenever possible to use graph patterns whose path passes through the statement nodes (e.g. example Query 2) rather than directly through truthy edges.

Conversion from CSV to RDF. The system described above has been used to manage four datasets that are subgraphs of the Wikidata knowledge graph. Table 2 summarizes conversion 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-tabular* was done under macOS 11.0.1 using a 2.3 GHz i5 (I5-8259U) quad core processor with 8 GB VRAM memory with data stored on an SSD drive.

The key point of these data is that converting the CSV files into RDF can take a substantial amount of time, even for relatively small datasets under a million triples. 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. It is also possible that the conversion time could be reduced if it were done using a more efficient application than *rdf-tabular*.

Compared to the time required to convert the CSVs to Turtle, the time required for Fuseki 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.

5. 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 [17]. As a W3C Recommendation, the CSV2RDF Recommendation fully discloses how the JSON 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 [16], and 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 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 JSON 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.

6. Conclusions

The W3C *Generating RDF from Tabular Data on the Web* Recommendation provides a mechanism for mapping CSV spreadsheets to the Wikibase model. When a CSV file is coupled with a JSON 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. That same mapping file contains sufficient information for software to be able to upload new data in the CSV file to the Wikidata API.

A system based on CSV files makes it easy for non-technical users to view and edit data. It allows them to use familiar spreadsheet software to edit data using built-in tools such as copy-and-paste and find-and-replace. They can more easily scan for patterns and errors by examining columns of data than they could using other serialization formats like JSON or XML.

Our experience using the system confirms that it can be used successfully by persons with little technical expertise. Despite having a full-time staff of two and no dedicated IT staff, the Vanderbilt Fine Arts Gallery was able to create items in Wikidata for over 6000 works in its collection.[30] Library staff with no experience or knowledge of Linked Data were able to review CSV data about journal titles for missing information and errors. And in two workshops, non-programmers were able to learn about, set up, and use the system to write data to Wikidata in less than an hour.

The simplicity of CSV data coupled with a JSON metadata description file makes it straightforward to store versioned snapshots of small subgraphs of Wikidata using a standard archival format. At any time, those snapshots can be transformed into RDF and loaded into a triplestore. Then 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. Those queries on a restricted subgraph of Wikidata provide a simple alternative to schema- or machine learning-based alternatives for detecting vandalism.

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

[30] https://www.wikidata.org/wiki/Wikidata:WikiProject_Vanderbilt_Fine_Arts_Gallery/Property_Coverage

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 GLAM institutions and other 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-tabular application. Thanks to Mark Denning for assistance with the web tool for constructing JSON metadata description files. Jakub Klimek, John Samuel, Tom Baker, Andra Waagmeester, and Dimitris Kontokostas provided helpful comments and suggestions during the review process that significantly improved the paper.

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
```

JSON metadata description file (filename `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-tabular` command line to emit RDF Turtle (installation of the `linkeddata` gem required for Turtle output):

```
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
```

JSON metadata description file (filename `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
```

JSON metadata description file (filename `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/>
```

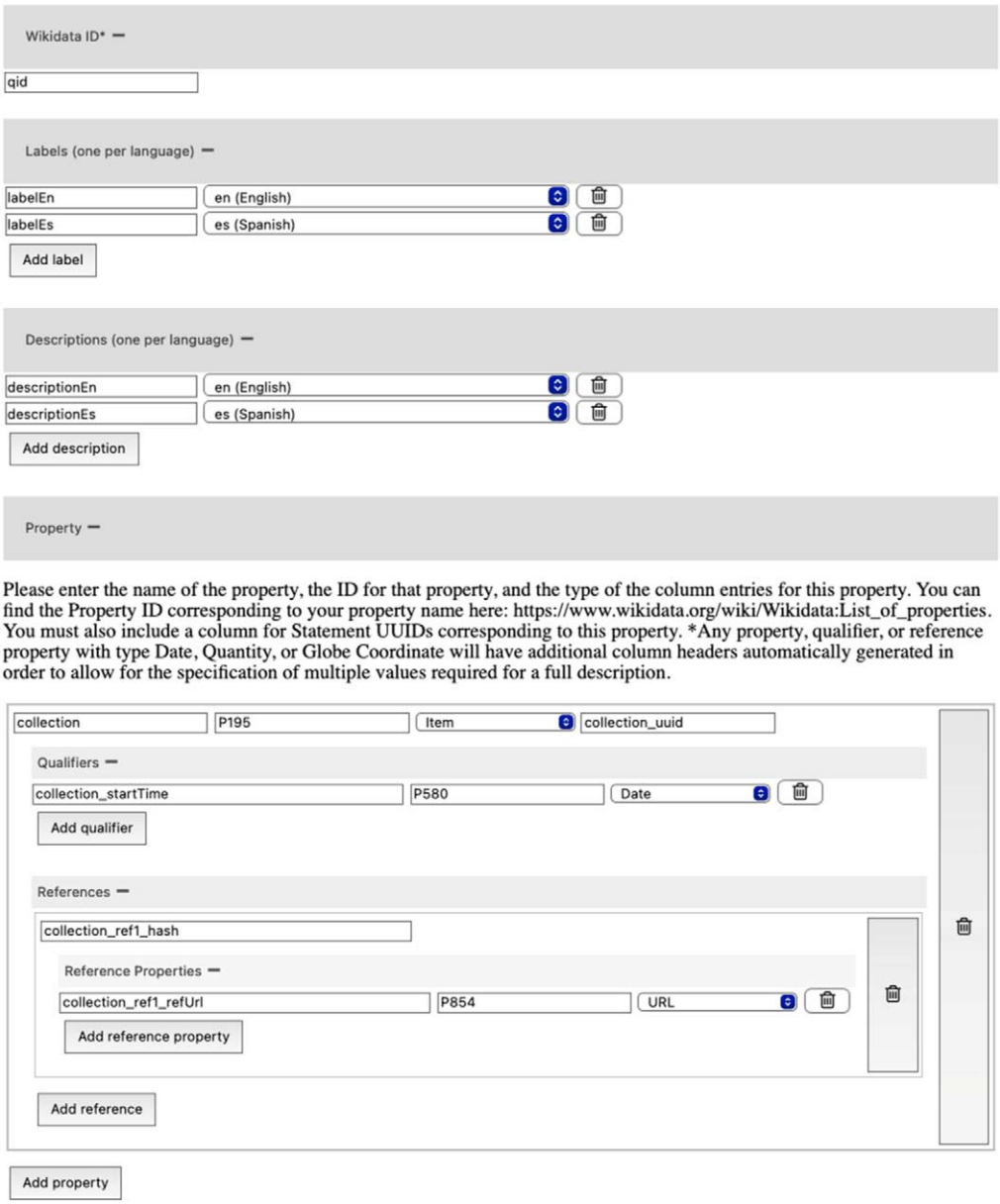
```
prefix  psv:  <http://www.wikidata.org/prop/statement/value/>
'''

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')
```

Appendix B. Web tool for constructing JSON metadata description files based on the Wikibase model

An operating version of the tool is available at https://heardlibrary.github.io/digital-scholarship/script/wikidata/wikidata-csv2rdf-metadata.html

The tool can be operated offline by downloading the HTML file https://github.com/HeardLibrary/linked-data/blob/master/vanderbot/wikidata-csv2rdf-metadata.html and its associated .js and .css files, then opening it in a browser.



References

[1] G. Antoniou, S. Batsakis, A. Isaac, A. Scarnhorst, M. García, R. van Horik, D. Giaretta and C. Meghini, Analysis of the limitations of digital preservation solutions for preserving linked data. PRELIDA (preserving linked data) ICT-2011.4.3: Digital preservation, 2014, https://openportal.isti.cnr.it/data/2014/308483/2014_308483.pdf [Accessed 1 June 2021].

[2] N. Arndt, P. Naumann, N. Radtke, M. Martin and E. Marx, Decentralized collaborative knowledge management using git, *Journal of Web Semantics* **54** (2018), 4–28. doi:10.2139/ssrn.3248491.

[3] Association of Research Libraries Task Force on Wikimedia and Linked Open Data, ARL white paper on Wikidata, 2019, https://www.arl.org/resources/arl-whitepaper-on-wikidata/ [Accessed 1 June 2021].

[4] T. Baker and E. Prud'hommeaux (eds), *Shape Expressions (ShEx) 2.1 Primer*, Shape Expressions Community Group, http://shex.io/shex-primer/.

[5] M. Bartoli, F. Guernaccini and G. Michetti, Preservation of linked open data, *JLIS.it* **11**(2) (2020), 20–44. doi:10.4403/jlis.it-12633.

[6] S. Baskauf, É.Ó. Tuama, D. Endresen and G. Hagedorn (eds), *Report of the TDWG Vocabulary Management Task Group (VoMaG) v1.0. Biodiversity Information Standards (TDWG)*, 2013, http://www.gbif.org/document/80862 [Accessed 1 June 2021].

[7] Biodiversity Information Standards (TDWG) Technical Architecture Group, TDWG technical roadmap, 2008, https://github.com/tdwg/rdf/blob/master/history/TAG_Roadmap_2008.pdf [Accessed 1 June 2021].

[8] D. Brickley, R. Guha and B. McBride (eds), *RDF Schema 1.1 W3C Recommendation, 25 February 2014*, http://www.w3.org/TR/rdf-schema/.

[9] R. Cyganiak, D. Wood and M. Lanthaler (eds), *RDF 1.1 Concepts and Abstract Syntax W3C Recommendation, 25 February 2014*, https://www.w3.org/TR/rdf11-concepts/#section-skolemization.

[10] J. Debattista, J. Fernández, M.-E. Vidal and J. Umbrich, Managing the evolution and preservation of the data web, *Journal of Web Semantics* **54** (2019), 1–3. doi:10.1016/j.websem.2018.12.011.

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

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

[13] S. Harris, A. Seaborne and E. Prud'hommeaux, PARQL 1.1 query language, 21 March 2013, https://www.w3.org/TR/sparql11-query/.

[14] R. Hyam, Managing the managing of the TDWG ontology, http://www.hyam.net/blog/archives/643 [Accessed 1 June 2021].

[15] LD4-Wikidata Affinity Group, https://www.wikidata.org/wiki/Wikidata:WikiProject_LD4_Wikidata_Affinity_Group [Accessed 2 September 2021].

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

[17] 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].

[18] B. Lóscio, C. Burle and N. Calegari (eds), *Data on the Web Best Practices W3C Recommendation, 10 January 2017*, https://www.w3.org/TR/dwbp/.

[19] S. Malyshev and G. Lederrey, Wikibase/indexing/RDF dump format, https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format [Accessed 22 December 2020].

[20] Meetup/cost MOBILISE Wikidata workshop, 2020, https://en.wikipedia.org/wiki/Wikipedia:Meetup/Cost_MOBILISE_Wikidata_Workshop [Accessed 22 December 2020].

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

[22] Object Management Group, Life sciences identifiers specification, OMG final adopted specification, May 2004, https://www.omg.org/cgi-bin/doc?dtc/04-05-01.

[23] Organization for Standardization, Space data and information transfer systems – open archival information system (OAIS) – reference model (second edition), ISO 14721:2012(E), 2012, https://www.iso.org/standard/57284.html.

[24] A. Sarabadani, A. Halfaker and D. Taraborelli, Building automated vandalism detection tools for Wikidata, in: *WWW'17 Companion: Proceedings of the 26th International Conference on World Wide Web Companion*, 2017, pp. 1647–1654. doi:10.1145/3041021.3053366.

[25] R. Taelman, M. Sande, J. Herwegen, E. Mannens and R. Verborgh, Triple storage for random-access versioned querying of RDF archives, *Journal of Web Semantics* **54** (2018), 29–47. doi:10.2139/ssrn.3248501.

[26] 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/.

[27] Taxonomic Databases Working Group (TDWG) Technical Architecture Group, Technical roadmap, 2006, https://github.com/tdwg/rdf/blob/master/history/TAG_Roadmap_2006.pdf [Accessed 1 June 2021].

[28] 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/.

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

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

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

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

[33] Wikimedia Foundation, Wikidata:Glossary, https://www.wikidata.org/wiki/Wikidata:Glossary [Accessed 22 December 2020].
[34] Wikimedia Foundation, Wikibase JSON format, https://doc.wikimedia.org/Wikibase/master/php/md_docs_topics_json.html [Accessed 2 September 2021].
[35] Wikimedia Germany, Welcome to the Wikibase project, https://wikiba.se/ [Accessed 22 December 2020].