The Wikibase Bulk Kit

Roberto Alvarado[1], Aidan Hogan[1]

[1]DCC, Universidad de Chile & Instituto Milenio Fundamentos de los Datos

Abstract

In this resource paper, we present the Wikibase Bulk Kit (WBK) framework to facilitate the extraction, modelling and import of legacy data into Wikibase instances. WBK accepts one or more input data files (currently CSV format is supported), declarative configuration & mapping files, and a Wikibase instance, and imports the corresponding data files into Wikibase per the mapping files. A first declarative file configures a particular project. A second declarative specification defines the vocabulary and schema required for representing the data. A third file defines how the input file should be imported into Wikibase. Based on these specifications, and input data, a processor then populates a Wikibase instance while reconciling the new data with the existing entities and relations in the knowledge graph.

1. Introduction

Wikibase – the software platform that powers Wikidata – is becoming a popular platform for managing knowledge graphs. Its many features – including entity-centric browsing, keyword search and autocompletion features, multilingualism, support for multimedia, lightweight support for reification via qualifiers, external identifier schemes, rich type system, Linked Data exports, SPARQL query service and user-interface, collaborative editing platform, etc. – make it a fully-fledged solution in such scenarios. Aside from Wikidata itself [1], the Wikibase Ecosystem platform currently lists 328 total instances[1]; some of the most prominent include the Mathematical Research Data Initiative (MaRDi) portal [2], the EU Knowledge Graph [3], FactGrid [4], etc. The largest, the MaRDi portal, which supports "*handling mathematical research data*", contains 861,680,648 triples at the time of writing.

Despite the growing popularity of Wikibase as a platform for open knowledge graphs, there are still technical barriers-to-entry for adopting it in the context of such initiatives.

One established issue is that of bulk imports into Wikibase, where the off-the-shelf mechanisms load items very slowly, despite the fact that the underlying database technologies can process inserts orders of magnitude faster: Shigapov et al. [5] estimate an insertion rate of 1–6 items per second for the API, and 220–280 items per second via the underlying database. The slowdown is due to multitudinous API calls, internal validation, and transaction overhead, and is not solved by standard tools, such as QuickStatements[2] or the Wikidata Toolkit[3], which use the API for loading. While QuickStatements 3.0 does improve bulk performance by grouping

Wikidata'26: Wikidata Workshop at ISWC 2026

roberto.alvarado@ug.uchile.cl (R. Alvarado); ahogan@dcc.uchile.cl (A. Hogan)

0000-0001-9482-1982 (A. Hogan)

[1]See https://wikibase-metadata.wmcloud.org/; retr. 2026-07-30.

[2]https://quickstatements.toolforge.org/; retr. 2026-07-30

[3]https://github.com/Wikidata-Toolkit/Wikidata-Toolkit; retr. 2026-07-30

commands relating to a single entity into one API request, the documentation clarifies that "*large batch runs will take a long time to complete*". On the Wikimedia issue tracker, a ticket opened in 2021 for bulk imports remains stalled.[4] Third-party solutions such as RaiseWikibase help [5], but without maintenance, do not work for newer versions of Wikibase.

Another barrier-to-entry is the lack of declarative solutions to describe how legacy data should be imported into Wikibase. While QuickStatements provides a declarative way to describe and execute a variety of updates on Wikibase, it does not support bulk imports, and does not support mapping from legacy formats into Wikibase's data model.

In order to facilitate more projects with legacy data using Wikibase as a platform for open knowledge graphs, we see a need for – and call for collaboration on – a solution that:

1. allows for declaratively defining a schema in Wikibase, including the definition of class and property terms needed to describe instance data;
2. allows for declaratively mapping legacy data into the Wikibase model per the defined schema, with options for reconciling entities and relations with respect to existing items and statements, or creating new items and statements, as necessary;
3. supports high-throughput bulk imports and updates (per RaiseWikibase).

We propose the Wikibase Bulk Kit (WBK): a framework to address these three points.

In particular, WBK supports the specification of Wikibase schemata and mappings via a YAML-based format, addressing the first two points. Regarding the second point, the framework further supports a variety of features and settings for reconciliation, which aims to avoid creating duplicate items and statements in Wikibase by considering those that already exist. Currently the framework supports CSV-like formats but can be extended in future to other data formats. Regarding the third point, it supports two forms of updating Wikibase: via the API (with internal validation), and via bulk upload (supported via a patched version of RaiseWikibase).

The development of this framework is inspired by an ongoing project to create an open knowledge graph from Chile's open data portals. These portals primarily consist of tabular formats, with thousands of tables available. Early efforts involved coding the logic of imports and the mapping into scripts. WBK separates the logic for importing data into Wikibase from the definition of the mapping of legacy data, while also supporting bulk imports and reconciliation.

Herein we describe WBK, its design, the mappings it supports, and its current limitations.

2. Related Works

With respect to mapping legacy formats into knowledge graphs, a wide range of languages and tools have been proposed. In the context of RDF knowledge graphs, a seminal development in this direction was the publication of the R2RML standard for mapping relational databases to RDF graphs [6]. Dimou et al. [7, 8] later proposed to generalise this specification in order to support mapping a variety of legacy formats (CSV, JSON, etc.) to RDF graphs. An alternative approach to specify such mappings are the SPARQL-based languages, such as SPARQL Generate [9], SPARQL Anything [10], ShExML [13], etc. Though these works have had considerable impact,

[4] https://phabricator.wikimedia.org/T287164; 2026-07-30.

they focus on RDF as a target format. Though Wikibase has an RDF export, and includes a SPARQL query engine, it does not have a mechanism for importing from RDF, and hence – though we can take inspiration from such works – they are not directly applicable in our scenario. Furthermore, they do not incorporate reconciliation methods.

Another key framework for constructing knowledge graphs from legacy data is OpenRefine, which supports a variety of features for Wikidata and other Wikibase instances.[5] These features overlap with many requirements for WBK, including support for a variety of legacy formats, support for importing data into Wikibase, support for reconciling entities with existing items, etc. However, the tool is primarily intended for use via a graphical interface, whereas we aim for programmatic application of mappings. Furthermore, the tool does not support bulk imports. For these reasons, we opted for a more lightweight approach customised for Wikibase.

A number of solutions have been proposed for bulk imports to Wikibase based on inserting data directly into the underlying relational database as part of one large transaction. This approach avoids the overhead of sending multitudinous API requests, of managing transactions for individual updates, and of validating every update. Some limitations of such approaches are the lack of validation, and brittleness faced with changes to the relational database schema as Wikibase versions evolve. The most well-established tool in this space is RaiseWikibase [5].

With respect to these works, WBK is inspired by mapping languages that target RDF, but rather targets the Wikibase data model. It incorporates features found in OpenRefine, but does so in a declarative, programmatic way. Our work incorporates RaiseWikibase as a way to support bulk imports; this required patching the tool to make it compatible with the database schema and population defined for contemporary versions of Wikibase. We currently focus on support for input CSV files as required for our current projects, though WBK could be extended in future to support other formats, including JSON, XML, RDF, etc.

3. Wikibase Bulk Kit

We provide an overview of WBK in Figure 1, the engine of which is the Processor. Given a collection of legacy data as CSV files, and a Wikibase instance, the first step is to define some overall project settings. The second is to define the schema specification that will be used to represent the imported data. The Processor takes as input this specification and instructs Wikibase to insert all of the classes and properties that it specifies. Once this is completed, the third step is to define a mapping specification for each CSV file (CSVs with similar structure and content can reuse the same mapping specification). The Processor then takes each pair of CSV file and mapping specification and applies the corresponding operations on the Wikibase instance to import the legacy data as described by the specification. In this step, the mapping specification can indicate various semantics for the reconciliation of entities, and can also choose to conduct updates to Wikibase via either the API (slow, but validated by Wikibase) or via the bulk loader (fast, but not validated by Wikibase).

In the following, we present more details regarding the project specification, the schema specification, the mapping specifications, and how they are handled by the processor.

[5] https://openrefine.org/docs/manual/wikibase/overview; retr. 2026-07-30.

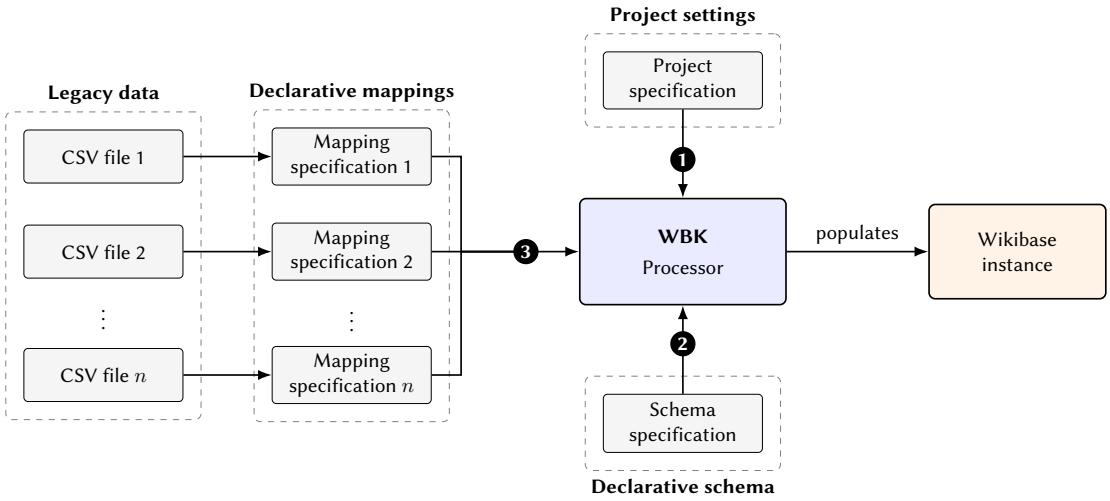

Figure 1: WBK overview

3.1. Declarative specifications

We will provide a running example to illustrate the main features of the three types of specifications: projects, schema and mapping. These examples will not cover all features supported, but in Figure 2, we provide an EBNF-style grammar that describes the complete YAML formats.

Project specification

The project specification provides the URLs for the Wikibase instance, for its MediaWiki API, for its SPARQL endpoint, and its username and password. Optionally, the connection details for the underlying MySQL database can be provided if RaiseWikibase is required.

Schema specification

The schema specification primarily defines the classes and properties required to model the data. It further permits static statements to be defined about them. We provide an example in Figure 3, which defines four properties and three items. Often the items defined in the schema specification will refer to classes. When creating properties and items, a label and description must be provided while aliases are optional. Properties must define their datatype, corresponding to those supported in Wikibase. Statements can also be provided on items.

Often it may be more intuitive to base schema statements on labels and/or descriptions, as shown on lines 33–39, where high school is defined as a subclass of school using labels. Referring to schema elements via their labels requires assuming that *labels are unique* specifically for schema elements (properties and items, typically representing classes). In case unique labels cannot be assumed, identifiers can be explicitly managed and specified for schema elements, where the aforementioned subclass statement would rather need to specify the identifiers P2 (subclass of) and Q1 (School) as defined previously in the specification.

Project specification

⟨*project*⟩ ::= ‘`wikibase_url:`’ ⟨*string*⟩
‘`mediawiki_api_url:`’ ⟨*string*⟩
‘`wikibase_username:`’ ⟨*string*⟩
‘`wikibase_password:`’ ⟨*string*⟩
(‘`sparql_endpoint_url:`’ ⟨*nullable-string*⟩)?
(‘`mysql_host:`’ ⟨*nullable-string*⟩)?
(‘`mysql_port:`’ ⟨*nullable-integer*⟩)?
(‘`mysql_database:`’ ⟨*nullable-string*⟩)?
(‘`mysql_user:`’ ⟨*nullable-string*⟩)?
(‘`mysql_password:`’ ⟨*nullable-string*⟩)?

⟨*nullable-string*⟩ ::= ⟨*string*⟩ | ‘`null`’

⟨*nullable-integer*⟩ ::= ⟨*integer*⟩ | ‘`null`’

Schema specification

⟨*schema*⟩ ::= (‘`namespace:`’ ⟨*string*⟩)?
(‘`language:`’ ⟨*string*⟩)?
‘`properties:`’ ⟨*property*⟩∗
‘`items:`’ ⟨*item*⟩∗

⟨*property*⟩ ::= ‘`- label:`’ ⟨*string*⟩
‘`description:`’ ⟨*string*⟩
‘`datatype:`’ ⟨*datatype*⟩
(‘`id:`’ ⟨*property-id*⟩)?
(‘`aliases:`’ ⟨*string-list*⟩)?

⟨*item*⟩ ::= ‘`- label:`’ ⟨*string*⟩
‘`description:`’ ⟨*string*⟩
(‘`id:`’ ⟨*item-id*⟩)?
(‘`aliases:`’ ⟨*string-list*⟩)?
(‘`statements:`’ ⟨*statement*⟩∗)?

⟨*statement*⟩ ::= ⟨*claim*⟩
(‘`qualifiers:`’ ⟨*claim*⟩∗)?
(‘`references:`’ ⟨*claim*⟩∗)?
(‘`rank:`’ ⟨*string*⟩)?

⟨*claim*⟩ ::= ‘`- value:`’ ⟨*string*⟩
‘`datatype:`’ ⟨*datatype*⟩
(‘`id:`’ ⟨*property-id*⟩)?
(‘`label:`’ ⟨*string*⟩)?

⟨*string-list*⟩ ::= ‘`[`’ (⟨*string*⟩ (‘`,`’ ⟨*string*⟩)∗)?‘`]`’

⟨*property-id*⟩ ::= ‘`P`’ ⟨*positive-integer*⟩

⟨*item-id*⟩ ::= ‘`Q`’ ⟨*positive-integer*⟩

⟨*datatype*⟩ ::= ⟨*string*⟩

⟨*mapping*⟩ ::= ‘`{`’ (⟨*string*⟩ ‘`:`’ ⟨*value*⟩) (‘`,`’ ⟨*string*⟩ ‘`:`’ ⟨*value*⟩)∗‘`}`’

⟨*value*⟩ ::= ⟨*string*⟩ | ⟨*number*⟩ | ⟨*boolean*⟩ | ‘`null`’ | ⟨*mapping*⟩

Mapping specification

⟨*mapping*⟩ ::= (‘`name:`’ ⟨*string*⟩)?
(‘`language:`’ ⟨*string*⟩)?
(‘`encoding:`’ ⟨*string*⟩)?
(‘`delimiter:`’ ⟨*string*⟩)?
(‘`decimal_separator:`’ ⟨*string*⟩)?
(‘`chunk_size:`’ ⟨*integer*⟩)?
‘`csv_files:`’ ⟨*csv-file*⟩+

⟨*csv-file*⟩ ::= ‘`- file_path:`’ ⟨*string*⟩
(‘`encoding:`’ ⟨*string*⟩)?
(‘`delimiter:`’ ⟨*string*⟩)?
(‘`decimal_separator:`’ ⟨*string*⟩)?
(‘`update_action:`’ ⟨*update-action*⟩)?
(‘`create:`’ ⟨*boolean*⟩)?
‘`mappings:`’ ⟨*mapping-rule*⟩+

⟨*mapping-rule*⟩ ::= ‘`- item:`’ ⟨*item-definition*⟩
(‘`update_action:`’ ⟨*update-action*⟩)?
(‘`create:`’ ⟨*boolean*⟩)?
(‘`label:`’ ⟨*string*⟩)?
(‘`description:`’ ⟨*string*⟩)?
(‘`statements:`’ ⟨*statement*⟩∗)?

⟨*item-definition*⟩ ::= ‘`label:`’ ⟨*string*⟩
(‘`description:`’ ⟨*string*⟩)?
(‘`snak:`’ ⟨*snak-matcher*⟩)?

⟨*snak-matcher*⟩ ::= ‘`property:`’ ⟨*string*⟩
‘`value:`’ ⟨*string*⟩

⟨*statement*⟩ ::= ‘`- property:`’ ⟨*string*⟩
‘`value:`’ ⟨*value-spec*⟩
(‘`qualifiers:`’ ⟨*statement*⟩∗)?
(‘`references:`’ ⟨*statement*⟩∗)?
(‘`rank:`’ ⟨*string*⟩)?

⟨*value-spec*⟩ ::= ⟨*scalar*⟩ | ⟨*value-definition*⟩ | ⟨*list*⟩ | ⟨*mapping*⟩

⟨*value-definition*⟩ ::= (‘`label:`’ ⟨*string*⟩)?
(‘`snak:`’ ⟨*snak-matcher*⟩)?

⟨*update-action*⟩ ::= ‘`append_or_replace`’ | ‘`force_append`’ | ‘`keep`’ | ‘`replace_all`’ | ‘`merge_refs_or_append`’ | ‘`merge_qualifiers_or_append`’

⟨*scalar*⟩ ::= ⟨*string*⟩ | ⟨*number*⟩ | ⟨*boolean*⟩ | ‘`null`’

⟨*list*⟩ ::= ‘`[`’ (⟨*value-spec*⟩ (‘`,`’ ⟨*value-spec*⟩)∗)?‘`]`’

⟨*mapping*⟩ ::= ‘`{`’ (⟨*string*⟩ ‘`:`’ ⟨*value-spec*⟩ (‘`,`’ ⟨*string*⟩ ‘`:`’ ⟨*value-spec*⟩)∗)?‘`}`’

⟨*boolean*⟩ ::= ‘`true`’ | ‘`false`’

Figure 2: Grammars for the project, schema, and mapping specifications.

Mapping specification

The purpose of a mapping specification is to indicate how to load input data in CSV format into the Wikibase instance according to the schema previously defined. Take the example tabular data shown in Table 1. For each row, we wish to (1) create a new item for each school with the corresponding label; (2) link each such new item via the *instance of* property to the item for the

```yaml
namespace: "education"
language: "en"

# Define items representing properties
properties:
 - label: "instance of"
   description: "Class this is an instance of"
   datatype: "wikibase-item"
   id: "P1"

 - label: "subclass of"
   description: "All instances are also instances of"
   datatype: "wikibase-item"
   id: "P2"

 - label: "located in"
   description: "Place in which an entity is located"
   datatype: "wikibase-item"

 - label: "founded"
   description: "Date the school was founded"
   datatype: "time"

# Define items representing classes
items:
 - label: "Neighbourhood"
   description: "Urban neighbourhood."

 - label: "School"
   description: "Educational institution"
   id: "Q1"

 - label: "High school"
   aliases: [ "Secondary school" ]
   description: "Institution for secondary education"
   statements:
    - value: "School"
      datatype: "wikibase-item"
      label: "subclass of"
```

Figure 3: Example of a schema specification

High school class; (3) reconcile each neighbourhood to an existing item or create one if it does not exist and link the school to this item via the *located in* property; (4) add a statement with the *founded* property indicating the year as a time value.

We present a mapping for this in Figure 4, which begins with some metadata and general settings for the mapping, and the inputs, including the in-scope language. Here one file is specified, but multiple with similar schema can be specified, in which case the union of the tables is used. The mapping starts with defining the items, specifically the neighbourhoods and the schools listed in the table. The table is processed row-by-row, and the mapping is applied to each row. We use braces of the form `{column}` to indicate a placeholder that will be replaced with the value of the corresponding column in the current row, potentially as part of a larger value that might include static strings, other placeholders, etc. Setting `create` to `true` indicates that if an item with the corresponding label/description is not found, it will be created along with the corresponding label. Of note is the value for the date in which the

Table 1
Example CSV file containing school information.

name	type	neighbourhood	founded
Instituto Nacional	High school	Santiago Centro	1813
Liceo N. 1 Javiera Carrera	High school	Santiago Centro	1894
Liceo José Victorino Lastarria	High school	Providencia	1913
…	…	…	…

school was founded providing the arguments required for creating such values in Wikibase: the first indicates the time string with the year placeholder; the second indicates the timezone (0 indicates Z or UTC); the third indicates the level of precision (9 indicates a year-level precision); and the fourth indicates the Wikidata ID of the calendar (Gregorian).

Regarding the language, we assume that CSV files will tend to be monolingual, and hence that one language applies per mapping. In case of multilingual files, where columns provide values in different languages, we currently require separate mapping specifications.

Though not shown in this example, support for qualifiers, references, ranks, snaks, etc., is provided in these mappings. Furthermore, in the context of legacy or fresh Wikibase instances, different reconciliation semantics can be chosen, as will be discussed in the following.

3.2. WBK Processor

The Processor is responsible for receiving the input data along with the three types of specifications, and using them to populate the Wikibase instance. The WBK Processor is implemented in Python, where the specifications are handled using `pydantic`, and the legacy tabular data are handled using `pandas`. We mention two non-trivial features of its implementation.

Reconciliation

Reconciliation refers to integrating updates with the existing elements of the Wikibase instance. Entity reconciliation ensures that each mention of a particular entity in the legacy data is represented by a coherent item, while relation reconciliation defines how new statements being generated from the legacy data should affect the existing statements in the Wikibase instance.

With reference to Table 1, as an example of entity reconciliation, given a fresh Wikibase instance, when we process the first row and first encounter the Santiago Centro neighbourhood, we may need to create a new item; on the other hand, when processing the second row, we want to avoid creating a duplicate item, and rather reuse the identifier for the existing item. Conversely, let's assume we are working with a Wikibase instance (e.g.) already populated from prior CSVs, and that prior to processing this particular CSV file, it specifies that Instituto Nacional is a primary school. Presented with the new fact that it is a high school, should we overwrite the previous value, skip, or add high school as a new value for instance of?

We support reconciling entities via a snak, i.e., the value of a property specified in the mapping, which can be considered an identifier for the entity. In the case of legacy data without explicit identifiers, we typically rely on the creation of unique entity descriptions potentially combining

```yaml
name: "Santiago high schools"
language: "es"
encoding: "utf-8"
delimiter: ","

csv_files:
 - file_path: "data/schools.csv"

mappings:
 # Reconcile neighbourhoods by label; create missing ones.
 - item:
   label: "{neighbourhood}"
   create: true
   statements:
    - property: "instance of"
      value: "Neighbourhood"

 # Create or reconcile schools by their names.
 - item:
   label: "{name}"
   create: true
   statements:
    - property: "instance of"
      value: "High school"

    - property: "located in"
      value:
      label: "{neighbourhood}"

    - property: "founded"
      value:
       - "+{founded}-00-00T00:00:00Z"
       - 0
       - 9
       - "http://www.wikidata.org/entity/Q1985727"
```

Figure 4: Example of a mapping specification

values from multiple columns. In the context of Table 1, if multiple schools have the same name, we recommend defining the label as "`{name}`", and the (disambiguating) description as "`{name} (in {neighbourhood})`", or "`{name} (est. {founded})`", or whatever combination might establish a (pseudo) primary key. Existing items, if present, are then found via the SPARQL endpoint (part of any standard Docker Wikibase instance). Sending several SPARQL queries for each row has a prohibitive cost. Thus the mapping can specify a chunk-size c, where it collects together c *unique* entity searches that it resolves with a single SPARQL query; this further removes duplicates within the chunk, improving efficiency.

Relation reconciliation is technically simpler, but conceptually more complex, given the intricacies of statements in Wikibase, and the context of the input data, the Wikibase instance, etc. For this reason, we allow the user to specify what is most appropriate in the specification. In Table 2, we provide the potential options. For example, an option like REPLACE_ALL might be appropriate when a new improved version of a particular input file has been released. On the other hand, KEEP might be appropriate when a new source is being considered.

Update strategy

We consider two update strategies that have distinctive strengths and weaknesses: API updates, and RaiseWikibase updates. API updates use the standard update mechanisms, which also incorporate various validation steps, and are maintained by the Wikibase team, but can have prohibitively slow update rates for importing large volumes of data. On the other hand, RaiseWikibase is a third-party extension that directly modifies the underlying relational database that, all going well, gives the same result; however, it skips validation steps, and it is not maintained with respect to new Wikibase versions. Hence RaiseWikibase is an excellent (and sometimes the *only*) choice for bulk imports at large scale, but can lead to importing "corrupted" data that the higher layers of Wikibase cannot parse or read. However, by creating the database insertions via WBK, we can require data to correspond to valid database instances that Wikibase recognises.

Assuming a relatively low volume of updates, we apply schema creation via the official API update mechanisms. For mappings, we rather use RaiseWikibase given that they may generate a large number of updates. Indeed, the latter is no longer maintained, was archived in 2024, and does not work for contemporary versions of Wikibase. Hence it required patching in order to work with recent versions due to changes in how the database is defined and populated [11].

4. Preliminary Evaluation

WBK was motivated by an ongoing project with students of the University of Chile to develop a knowledge graph for open data in Chile using Wikibase. The first prototype focused on educational data available on the Open Data Platform, and involved custom scripts written in Java, combining both the processing, and the mapping itself [12]. In this stage, we detected issues regarding (very) slow imports via the API [12]. The second stage thus explored integrating RaiseWikibase into the project, enabling the import of larger-scale data into the knowledge graph with orders of magnitude more efficiency [11]. The third stage considered, as described herein, the declarative mapping of the particular data source from the logic of the processor with the goal of starting to import a much broader swathe of data from the portal.

As a preliminary evaluation, we compare the ingestion performance for WBK and the ad hoc solution using the official API. The repository of source code of WBK is available at https://github.com/Hcatmin/wikibase-bulk-kit/.

Experimental setting The experiments were run on a machine with a 6-core Intel Core i5-11400H (11th Gen), 15 GB of RAM (9.4 GB used), an NVMe hard-disk, Ubuntu 24.04.3 LTS (Noble), and wikibase/wikibase-bundle:1.41.1-wmde.20 (MediaWiki 1.41.1).

Input data The input data were taken from the Open Data platform of the Ministry of Education in Chile.[6] The data sources used were: (1) The directory of educational establishments 2024, with 16,694 rows, and 15 Wikibase statements per row;[7] (2) Registration summary 2020–2023, with 73,314 rows, and 15 Wikibase statements per row;[8] (3) Summary of academic performance

[6] https://datosabiertos.mineduc.cl/; retr. 2026-07-31.
[7] https://datosabiertos.mineduc.cl/wp-content/uploads/2024/11/Directorio-Oficial-EE-2024-.rar; retr. 2026-07-31.
[8] https://datosabiertos.mineduc.cl/resumen-de-matricula-por-establecimiento-educacional/; retr. 2026-07-31.

Table 2
Configurations for relation reconciliation semantics

REPLACE_ALL	Deletes all existing statements from the item and reconstructs them entirely from the mapping.
APPEND_OR_REPLACE	If the item contains a statement with the same mainsnak, that statement is replaced; otherwise, it is added as a new statement.
FORCE_APPEND	Adds all new statements without checking for duplicates. This allows multiple claims with the same value to coexist.
KEEP	Preserves all current statements and adds only those properties that are not yet present on the item. Existing claims are not modified.
MERGE_REFS_OR_APPEND	If a claim exists with the same mainsnak and the same qualifiers, its references are merged; otherwise, a new claim is added.
MERGE_QUALIFIERS_OR_APPEND	Merges or extends the qualifiers of an existing claim when the mainsnak matches; otherwise, a new statement is added.

per institute 2020–2024, with 66,156 rows, and 5 Wikibase statements per row.[9]

Configuration Chunk size was set to 2,000 (whereby 2,000 entities are collected and potentially reconciled in a single SPARQL query). A fresh Wikibase instance was used.

Performance results The original ad-hoc scripts using Java updating Wikibase via the official API (without batching) process 0.139 rows per second on average across the three sources. Loading the entire dataset takes 327 hours. WBK configured to use RaiseWikibase processes 73.2 rows per second, taking 0.59 hours. These orders of magnitude speed-ups are in large part thanks to RaiseWikibase, and are consistent with the results already reported [5].

Among the three different sources, using WBK, the directory can be loaded with a rate of 56 rows per second, registration can be loaded with 73 rows per second, and performance can be loaded with 100 rows per second. This shows that the performance of loading legacy data depends not only on the number of statements inserted per row, but also the costs of reconciliation (for example, columns with repetitive values have less cost).

5. Discussion

Wikibase is a promising platform for supporting the creation, curation, management, interlinking and consumption of open knowledge graphs. However, we believe that there are key pieces needed to broaden its adoption. The first is the efficient ingestion of legacy data. The second is a declarative solution for mapping legacy data into a Wikibase instance.

[9] https://datosabiertos.mineduc.cl/resumen-de-rendimiento-por-unidad-educativa/; retr. 2026-07-31.

In this paper, we described the Wikibase Bulk Kit. The development of this framework was inspired by local needs: the creation of an open knowledge graph for Chilean open data. We believe that the same requirements are repeated in other scenarios, and hence we are interested in exploring collaboration and support in maintaining, improving and extending WBK.

There are many dimensions along which to improve WBK. The first is to support other legacy formats beyond CSVs. A second is to support more complex functions and operators, such as to perform joins, support functions to transform raw cell values, etc. A third is to continue to improve performance and validation steps when bulk import is invoked. A fourth point is to enable continuous updates as legacy sources are updated. A fifth direction is to support further Wikibase features, such as quality constraints, shapes, etc.

One option is (of course) to extend and improve WBK itself. A quick solution for the first two points might be to add support for a particular RDF format, and combine WBK (with a performance cost) with existing solutions, like RML [7], ShExML [13], etc., to map this RDF format to Wikibase; this would also offer support for more advanced functions and operators.

Wikibase holds much potential as a knowledge graph platform, but better solutions are required to ensure that such knowledge graphs can be constructed from legacy data. We believe that WBK fills a gap in this setting, but in order to have real impact, it requires a community to collaborate on extending, maintaining and continuously improving the framework.

Acknowledgments

This work was funded in part by ANID – Millennium Science Initiative Program – Code ICN17_002. We thank Marcelo Valenzuela for generating the pilot version of this knowledge graph, and Álvaro Tobar for working on patching RaiseWikibase.

Declaration on Generative AI

Claude and ChatGPT were used in the development of the WBK software, and for assistance with the generation of initial versions of figures in LaTeX. It was used to spot and correct typos in the final version of this paper but was not otherwise used for paper writing.

References

[1] D. Vrandecic, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Commun. ACM 57 (2014) 78–85. doi:10.1145/2629489.

[2] M. Schubotz, E. Ferrer, J. Stegmüller, D. Mietchen, O. Teschke, L. Pusch, T. Conrad, Bravo MaRDI: A Wikibase Knowledge Graph on Mathematics, in: L. Kaffee, S. Razniewski, K. Alghamdi, H. Arnaout (Eds.), Proceedings of the Wikidata Workshop 2023 co-located with 22nd International Semantic Web Conference (ISWC 2023), Athens, Greece, November 13, 2023, volume 3640 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3640/paper3.pdf.

[3] D. Diefenbach, M. D. Wilde, S. Alipio, Wikibase as an Infrastructure for Knowledge Graphs: The EU Knowledge Graph, in: A. Hotho, E. Blomqvist, S. Dietze, A. Fokoue,

Y. Ding, P. M. Barnaghi, A. Haller, M. Dragoni, H. Alani (Eds.), The Semantic Web - ISWC 2021 - 20th International Semantic Web Conference, ISWC 2021, Virtual Event, October 24-28, 2021, Proceedings, volume 12922 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 631–647. URL: https://doi.org/10.1007/978-3-030-88361-4_37. doi:10.1007/978-3-030-88361-4_37.

[4] O. Simons, Keine Selbstverständlichkeit: Citizen Science auf der FactGrid Wikibase-Plattform, in: R. Smolarski, H. Carius, M. Prell (Eds.), Citizen Science in den Geschichtswissenschaften, Göttingen, 2023. URL: https://doi.org/10.14220/9783737015714.241. doi:10.14220/9783737015714.241.

[5] R. Shigapov, J. Mechnich, I. Schumm, RaiseWikibase: Fast Inserts into the BERD Instance, in: R. Verborgh, A. Dimou, A. Hogan, C. d'Amato, I. Tiddi, A. Bröring, S. Maier, F. Ongenae, R. Tommasini, M. Alam (Eds.), The Semantic Web: ESWC 2021 Satellite Events - Virtual Event, June 6-10, 2021, Revised Selected Papers, volume 12739 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 60–64. URL: https://doi.org/10.1007/978-3-030-80418-3_11. doi:10.1007/978-3-030-80418-3_11.

[6] S. Das, S. Sundara, R. Cyganiak, R2RML: RDB to RDF Mapping Language, https://www.w3.org/TR/r2rml/, 2012.

[7] A. Dimou, M. V. Sande, P. Colpaert, R. Verborgh, E. Mannens, R. V. de Walle, RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data, in: C. Bizer, T. Heath, S. Auer, T. Berners-Lee (Eds.), Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea, April 8, 2014, volume 1184 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2014. URL: https://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf.

[8] A. Dimou, M. Vander Sande, B. De Meester, P. Heyvaert, T. Delva, RDF Mapping Language (RML), https://rml.io/specs/rml/, 2022.

[9] M. Lefrançois, A. Zimmermann, N. Bakerally, A SPARQL extension for generating RDF from heterogeneous formats, in: E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler, O. Hartig (Eds.), The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part I, volume 10249 of *Lecture Notes in Computer Science*, 2017, pp. 35–50. URL: https://doi.org/10.1007/978-3-319-58068-5_3. doi:10.1007/978-3-319-58068-5_3.

[10] L. Asprino, E. Daga, A. Gangemi, P. Mulholland, Knowledge Graph Construction with a *Façade*: A Unified Method to Access Heterogeneous Data Sources on the Web, ACM Trans. Internet Techn. 23 (2023) 6:1–6:31. URL: https://doi.org/10.1145/3555312. doi:10.1145/3555312.

[11] A. E. Tobar Salazar, Extensión y mejora del grafo de conocimiento de Datos Abiertos de Chile, Engineering Degree Thesis, Universidad de Chile, 2025.

[12] M. A. Valenzuela Barrera, Una nueva plataforma para los datos abiertos de Chile basado en Wikibase, Engineering Degree Thesis, Universidad de Chile, https://repositorio.uchile.cl/handle/2250/204548, 2024.

[13] H. García-González, I. Boneva, S. Staworko, J. E. L. Gayo, J. M. C. Lovelle, ShExML: improving the usability of heterogeneous data mapping languages for first-time users, PeerJ Comput. Sci. 6 (2020) e318. URL: https://doi.org/10.7717/peerj-cs.318. doi:10.7717/PEERJ-CS.318.