

Generierung angepasster RDF-Dumps von Wikidata

Benno Fünfstück

June 21, 2016

Version: My First Draft

Technische Universität Dresden

Fakultät Informatik
Institut für Theoretische Informatik
Professur für Wissensbasierte Systeme

Bachelorarbeit

Generierung angepasster RDF-Dumps von Wikidata

Benno Fünfstück

<i>1. Reviewer</i>	Prof. Markus Kroetzsch Fakultät Informatik Technische Universität Dresden
<i>2. Reviewer</i>	Prof. Sebastian Rudolph Informatik Technische Universität Dresden
<i>Supervisors</i>	Prof. Markus Kroetzsch and Prof. Sebastian Rudolph

June 21, 2016

Benno Fünfstück

Generierung angepasster RDF-Dumps von Wikidata

Bachelorarbeit, June 21, 2016

Reviewers: Prof. Markus Kroetzsch and Prof. Sebastian Rudolph

Supervisors: Prof. Markus Kroetzsch and Prof. Sebastian Rudolph

Technische Universität Dresden

Professur für Wissensbasierte Systeme

Institut für Theoretische Informatik

Fakultät Informatik

01062 and Dresden

Abstract

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: Dies ist ein Blindtext oder Huardest gefburn? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie Lorem ipsum dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Abstract (different language)

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: Dies ist ein Blindtext oder Huardest gefburn? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie Lorem ipsum dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Struktur der Arbeit	2
2	Hintergrund	4
2.1	Wikidata als strukturiertes Wiki	4
2.2	Wikidata als Linked Data	6
3	Anforderungen	11
3.1	funktionale Anforderungen	11
3.2	nicht-funktionale Anforderungen	12
3.3	Verwandte Arbeiten	13
4	Design	15
4.1	Architektur	15
4.2	Interface	18
4.3	Integration in existierende Infrastruktur	21
5	Implementierung	23
5.1	Datenmodell	23
5.2	Backend	25
5.3	Frontend	27
6	Evaluation	28
6.1	Vorgehen	28
6.2	Ergebnis	30
7	Fazit	33

Einleitung

Wikipedia ist die bekannteste freie Wissenssammlung im Internet. Bei fast 50 Millionen Artikeln in 278 Sprachen¹ Anfang 2019 sind Informationen zu einer Vielfalt an Themen verfügbar. Sogar YouTube und Facebook greifen auf Wikipedia zurück, um zusätzliche Informationen zu kontroversen Themen zu präsentieren.[@Gri18]

Die Darstellung der Informationen in natürlicher Sprache als Textartikel ist für Menschen praktisch. Doch für maschinelle Verarbeitung ist diese unstrukturierte Form der Darstellung nicht so gut geeignet, zumal sich die Fakten je nach Sprachversion unterscheiden können. Um diesem Problem zu entgegnen wurde 2012 das Wikidata Projekt gegründet, mit dem Ziel, die Informationen aus Wikipedia in atomare Aussagen zerlegt strukturiert zu verwalten[VK14]. Genau wie Wikipedia selbst kann auch Wikidata ohne Beschränkungen frei bearbeitet werden.

Ein Beispiel für die Datenrepräsentation in Wikidata zeigt Abb. 1.1. Mit Wikidata ist es möglich, auch verschiedene zunächst widersprüchlich erscheinende Fakten parallel darzustellen. Abhängig von der Art der Bestimmung (*determination method*) können entweder der Mount Everest oder auch Chimborazo als höchster Punkt der Erde gesehen werden. Diese Flexibilität ist notwendig, weil Wikidata ein Community-Projekt ist und deswegen auch die Möglichkeit haben muss, Fakten differenziert abzubilden um Konsens zu erreichen. Wie man im Beispiel sieht, können Statements zusätzlich auch noch mit Referenzen untermauert werden. Um Mehrdeutigkeiten zu vermeiden, werden alle Entitäten in Wikidata durch eindeutige IDs referenziert, und Statements verlinken auf diese IDs.

Für viele Projekte sind die Daten aus Wikidata aufgrund dieser Eigenschaften besonders interessant. So können die IDs zur eindeutigen Unterscheidung verschiedener Konzepte dienen und gleichzeitig noch weitere mit diesen Konzepten verlinkte Informationen abgerufen werden. Auch die Mehrsprachigkeit der Daten ist zur Zuordnung von gleichen Konzepten unterschiedlicher Sprachen sehr nützlich. Der kollaborative Aspekt von Wikidata wirft weitere interessante Fragestellungen, zum Beispiel zur Qualität der Daten[Bra+16] oder der Verwendung von Referenzen[Pis+17b][Pis+17a] auf.

¹<https://stats.wikimedia.org/DE/TablesArticlesTotal.htm>

<u>highest point</u>	<u>Mount Everest</u>	
	<u>elevation above sea level</u>	8,848±20 metre
	<u>determination method</u>	<u>sea level</u>
	0 references	
	<u>Chimborazo</u>	
	<u>determination method</u>	<u>geographical center</u>
	<u>length</u>	6,384.41598 kilometre
	1 reference	
	<u>reference URL</u>	https://www.theweathernetwork.com/us/news/articles/climate-and-environment/ecuadors-mt-chimborazo-is-officially-highest-spot-on-earth/66219

Abbildung 1.1: Zwei Statements des Items “Erde” für die Property “höchster Punkt”

Wikidata bietet bereits mehrere Wege für den Zugang zu den Daten an. Für einfache Abfragen zu einzelnen Objekten gibt es die Wikidata API, für komplexere Abfragen die auch Relationen zwischen mehreren Entitäten nutzen, bietet sich der Wikidata Query Service an. Außerdem werden Datenexports sowohl in einem Wikidata eigenen JSON-basierten Format sowie als RDF angeboten.

Doch durch das Wachstum von Wikidata sind neue Methoden zum Zugang zu den Daten notwendig. Mittlerweile besteht Wikidata aus mehr als 700 Million Statements zu fast 60 Millionen Items. Die vollständigen RDF-Datenexporte sind damit selbst komprimiert über 50 GB groß, und der Wikidata Query Service produziert Timeouts nicht aufgrund der Komplexität der Anfrage, sondern weil zu viele Daten abgefragt werden.[Pis+17b]. Es fehlt also ein System für den Bereich zwischen Abfragen kleinerer Datenmengen (Wikidata API, Wikidata Query Service) und dem Verarbeiten vollständiger Exporte. Als Lösung für dieses Problem wird in dieser Arbeit ein System vorgestellt, welches nach Angabe von Filterkriterien Dumps mit einer Teilmenge der Daten generieren kann.

1.1 Struktur der Arbeit

Im zweiten Kapitel wird zunächst notwendiges Hintergrundwissen zum Aufbau von Wikidata und den verwendeten Technologien vermittelt. Danach werden in Kapitel 3 die Anforderung an das System detaillierter beschrieben und verwandete Arbeiten

verglichen. In Kapitel 4 wird dann das Design der System erarbeitet, dessen Implementierung in Kapitel 5 vorgestellt wird. In der folgenden Evaluation in Kapitel 6 wird die Implementierung auf Erreichen der Anforderung geprüft und auch ein Vergleich mit den existierenden Wikidata-Dumps vorgenommen. Im letzten Kapitel ?? wird schließlich auf mögliche zukünftige Verbesserungen eingegangen, bevor ein Fazit gefällt wird.

Dabei werden die folgenden Beiträge geleistet:

- Analyse verschiedener Systemdesigns zur Generierung angepasster Wikidata-Dumps
- Entwurf einer Spezifikation für Filterkriterien
- Implementierung des vorgestellten Designs mit Wikidata-Toolkit
- Evaluation der Vollständigkeit und Korrektheit des RDF-Exports in Wikidata-Toolkit

Hintergrund

In diesem Kapitel werden Grundlagen zu Wikidata und RDF¹ (Resource Description Framework), einem Standardformat zur Beschreibung von Informationen, erklärt.

2.1 Wikidata als strukturiertes Wiki

Wikidata ist die gemeinsame Wissensdatenbank der Wikimedia Projekte. Wie auch Wikipedia selbst baut Wikidata auf MediaWiki² auf, einer Software für das Betreiben von kollaborativen Wikis. Im Gegensatz zu Wikipedia verwaltet Wikidata jedoch strukturierte Dokumente. Die Erweiterungen für MediaWiki dazu stellt das Wikibase³ Projekt bereit.

Alle Dokumente in Wikidata besitzen einen eindeutigen Identifier. Dieser beginnt mit einem Großbuchstaben, der den Typ des Dokuments angibt, gefolgt von einer Zahl. Aktuell kennt Wikidata drei Typen von Dokumenten: Items (Q), Properties (P) und Lexemes (L). Lexemes wurden erst später hinzugefügt, so dass sie von dem in dieser Arbeit verwendeten Wikidata-Toolkit noch nicht vollständig unterstützt werden. In dieser Arbeit werden Lexemes daher nicht genauer betrachtet.

Den Hauptbestandteil der Daten bilden jedoch die Items. Ein Item repräsentiert ein bestimmtes Konzept, zu dem Fakten in Wikidata erfasst werden. Der Aufbau eines Items ist beispielhaft anhand von Q42 (Douglas Adams) in Abb. 2.1 dargestellt. Den ersten Teil bilden die Terme: `label`, `description` und `aliases` dienen zur Beschreibung und Definition des Items. Die Terme sind mehrsprachig: ein Item kann ein `label` für jede der möglichen Sprachen haben. Darauf folgen Statements, welche die Fakten zu diesem Item wiedergeben. Zusätzlich kann jeder Item noch eine Reihe an Sitelinks besitzen. Diese verlinken auf andere Seiten des Wikimedia Projekts zum Thema des Items. Die Sitelinks von Q42 verlinken zum Beispiel auf Artikel von

¹<https://www.w3.org/RDF/>

²<https://mediawiki.org>

³<https://wikiba.se>

Douglas Adams in den unterschiedlichen Sprachversionen von Wikipedia und in Wikiquote.

Die Fakten eines Items werden in Statements beschrieben. Ein Statement besteht aus zwei Teilen: einem Claim, der eine bestimmte Aussage trifft, und einer Liste von Referenzen. Da auch Statements ohne Referenzen erlaubt sind, kann die Liste der Referenzen auch leer sein. Wikidata unterstützt drei Arten von Aussagen, genannt Snaks:

PropertyValue die am meisten verwendete Art einer Aussage. Sie beschreibt den Fakt, dass eine bestimmte Eigenschaft (Property) einen gewissen Wert (Value) hat. Die Property bestimmt dabei den Typ der Value. Values können je nach Property andere Items oder skalare Werte (wie Zahlen, Zeichenketten, Koordinaten, Zeiten, usw.) sein.

SomeValue diese Art der Aussage wird verwendet, um auszudrücken, dass ein Wert für die Eigenschaft existiert der aber nicht bekannt ist. Zum Beispiel kann die Aussage "es existiert ein Wert für den Todeszeitpunkt einer Person" verwendet werden, falls eine Person gestorben ist, der Todeszeitpunkt aber nicht bekannt ist.

NoValue drückt aus, dass es für eine bestimmte Eigenschaft keinen Wert gibt. Wird verwendet, wenn das Fehlen einer Information keine Unvollständigkeit darstellt. Kann zum Beispiel für P200 (Zuflüsse) verwendet werden, wenn ein Gewässer keine Zuflüsse besitzt.

Der Claim eines Statements besteht aus einem MainSnak, der die Hauptaussage darstellt, und zusätzlichen Qualifiern zur Verfeinerung der Aussage. Eine Referenz ist auch einfach eine Liste von Snaks. Ein MainSnak zu Douglas Adams (Q42) ist zum Beispiel P69 (educated at) - Q691283 (St John's College). Mit den Qualifier-Snaks wie P582 (end time) - 1974 bildet dieser dann einen Claim. Das Statement setzt sich dann aus diesem Claim und der Liste von Referenzen zusammen. Alle Statements für die gleiche Property werden in einer Statement Group zusammengefasst. Jedes Statement besitzt zusätzlich noch einen Rank (deprecated, normal oder best) um die Priorität innerhalb der Statement Group auszudrücken.

Dieses Dokument-orientierte Datenmodell lässt sich einfach als JSON⁴ (JavaScript Object Notation) repräsentieren. Die vollständige Darstellung ist jedoch sehr umfangreich und ist deshalb aus Platzgründen hier nicht abgebildet. Für ein Entity kann

⁴<https://www.json.org>

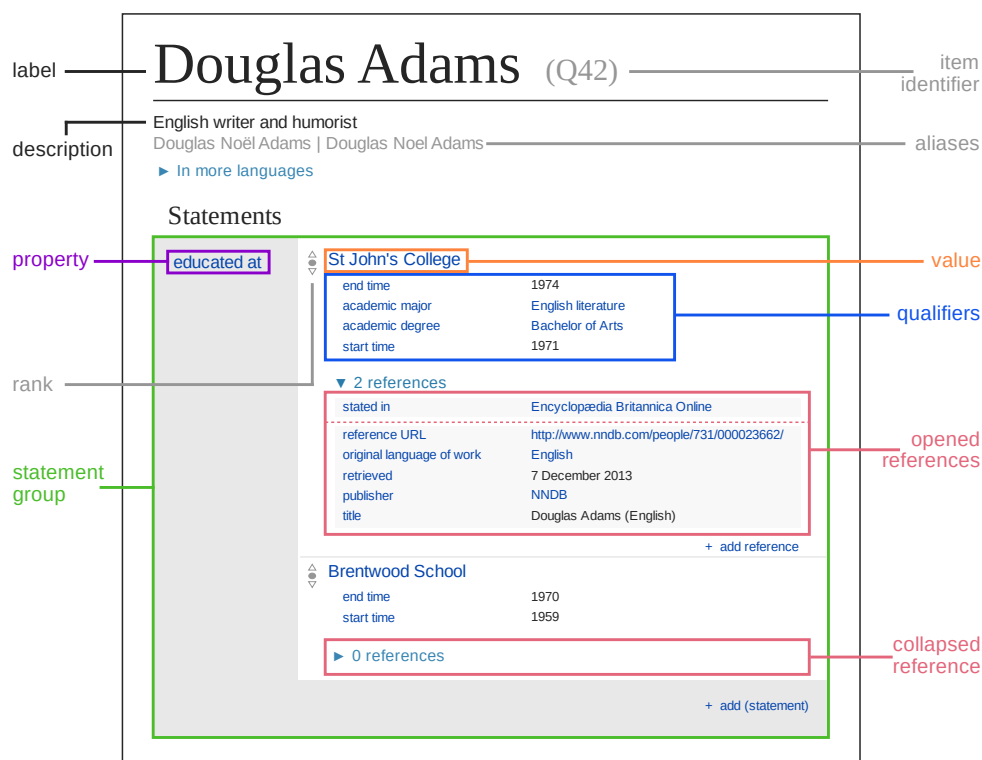


Abbildung 2.1: Wikidata Item “Q42”

die JSON-Repräsentation einfach online abgerufen werden, für Q42 zum Beispiel unter folgender URL: <https://www.wikidata.org/wiki/Special:EntityData/Q42.json>. Zusätzlich stellt Wikidata Exporte aller Dokumente als JSON bereit.

2.2 Wikidata als Linked Data

Eine Besonderheit von Wikidata ist die starke Verlinkung der Daten untereinander. Diese Verlinkung ermöglicht eine Art des Zugriffs, die sich von der dokumentbasierten Betrachtungsweise unterscheidet. Fragestellungen wie “Wer sind die Verwandten von Douglas Adams?” oder “Welche berühmten Personen sind in einem Staat geboren, der Mitglied der europäischen Union ist?” betrachten die Beziehungen der Dokumente, und nicht allein den Inhalt einzelner Dokumente.

Das W3C hat für solche Graph-basierten Daten das Resource Description Framework (RDF) entwickelt. In RDF werden als eindeutige Bezeichner Internationalized Resource Identifiers (IRIs)⁵ verwendet. IRIs sind eine Erweiterung von URIs zur Unterstützung von internationalen Zeichensätzen. In der Praxis werden für IRIs oft HTTP

⁵<https://www.ietf.org/rfc/rfc3987.txt>

URLs verwendet, sodass Namenskonflikte zwischen unterschiedlichen Organisationen vermieden werden. Außerdem können über HTTP weitere Informationen zu der entsprechenden Ressource bereitgestellt werden.

Das Kernelement von RDF bilden Tripel. Die drei Komponenten jedes Tripel sind:

Subjekt eine IRI oder Blank Node

Prädikat eine IRI

Objekt eine IRI, Blank Node oder Literal

Blank Nodes sind lokale Bezeichner, die im Gegensatz zu IRIs nur innerhalb eines Dokuments eindeutig sein müssen. Verschiedene RDF Dokumente können daher die selben Blank Node Bezeichner verwenden, ohne dass damit die selbe Resource beschrieben wird.

Literale in RDF sind Zeichenketten, die jedoch zusätzlich noch einen Datentypen oder einen Language Tag besitzen können. Mit dem Datentyp kann die Interpretation der Zeichenkette genauer spezifiziert werden. Auch Datentypen werden über IRIs referenziert. Language Tags geben die Sprache der Zeichenkette als Language Code nach BCP47⁶ an. In diesem Fall muss der Datentyp `http://www.w3.org/1999/02/22-rdf-syntax-ns#langString` sein.

RDF Dokumente sind eine Kollektion von Tripeln, wobei die Reihenfolge keine Rolle spielt. Für die Serialisierung von RDF gibt es verschiedene Standards, ein einfacher und verbreiteter ist N-Triples⁷. Die Syntax von N-Triples ist in folgendem Beispiel exemplarisch dargestellt:

```
1 <https://example.org> <http://schema.org/name> "Beispiel"@de .
2 <https://example.org> <http://schema.org/name> "Example"@en .
3 _:paper <https://example.org/about> <https://example.org> .
4 _:paper <https://example.org/popularity> "42"^^<http://www.w3.org/2001/
   ↪ XMLSchema#integer> .
```

Jede Zeile entspricht einem Tripel und wird mit einem Punkt abgeschlossen. In N-Triples werden IRIs in `<>` eingeschlossen und Blank Nodes durch das Prefix `_:` markiert. Literale sind von `"` umgeben, mit `@` bzw `^^` können Language Tag und Datentyp angegeben werden.

In dieser Arbeit wird zur Übersichtlichkeit eine Erweiterung der Notation verwendet. Dabei werden IRIs durch die Einführung von den in ?? aufgeführten Prefixen

⁶<https://tools.ietf.org/html/bcp47>

⁷<https://www.w3.org/TR/n-triples/>

Prefix	URL
rdf:	<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
xsd:	<http://www.w3.org/2001/XMLSchema#>
rdfs:	<http://www.w3.org/2000/01/rdf-schema#>
owl:	<http://www.w3.org/2002/07/owl#>
skos:	<http://www.w3.org/2004/02/skos/core#>
schema:	<http://schema.org/>
geo:	<http://www.opengis.net/ont/geosparql#>
prov:	<http://www.w3.org/ns/prov#>
wikibase:	<http://wikiba.se/ontology#>
wdata:	<http://www.wikidata.org/wiki/Special:EntityData/>
wd:	<http://www.wikidata.org/entity/>
wdt:	<http://www.wikidata.org/prop/direct/>
wdtn:	<http://www.wikidata.org/prop/direct-normalized/>
wds:	<http://www.wikidata.org/entity/statement/>
p:	<http://www.wikidata.org/prop/>
wdref:	<http://www.wikidata.org/reference/>
wdv:	<http://www.wikidata.org/value/>
ps:	<http://www.wikidata.org/prop/statement/>
psv:	<http://www.wikidata.org/prop/statement/value/>
psn:	<http://www.wikidata.org/prop/statement/value-normalized/>
pq:	<http://www.wikidata.org/prop/qualifier/>
pqv:	<http://www.wikidata.org/prop/qualifier/value/>
pqn:	<http://www.wikidata.org/prop/qualifier/value-normalized/>
pr:	<http://www.wikidata.org/prop/reference/>
prv:	<http://www.wikidata.org/prop/reference/value/>
prn:	<http://www.wikidata.org/prop/reference/value-normalized/>
wdno:	<http://www.wikidata.org/prop/novalue/>

Tabelle 2.1: Verwendete Prefixe. Diese werden auch vom Wikidata SPARQL Service vordefiniert.

vereinfacht. Zum Beispiel wird `<http://schema.org/name>` als `schema:name` abgekürzt.

RDF kennt weder Referenzen noch Qualifier und kann im Gegensatz zu Wikidata auch nicht mehrere gleiche Tripel speichern. Erxleben, Günther, Krötzsch, Mendez und Vrandečić haben 2014 beschrieben, wie trotzdem eine Darstellung als RDF möglich ist und auch ein System zur Erstellung regelmäßiger Exporte als RDF entwickelt [Erx+14]. Nach dem Prinzip der Reifikation werden Statements, Referenzen und komplexe Werte nicht direkt als Tripel exportiert, sondern als eigene Ressourcen. Diese Ressourcen besitzen dann ein Tripel das zu dem Objekt des Statements verlinkt, aber auch zusätzlich Tripel für Qualifier und Referenzen. Jedes Statement hat somit einen eindeutigen Namen, sodass auch zwei Statements mit dem selben Inhalt erfasst werden können.

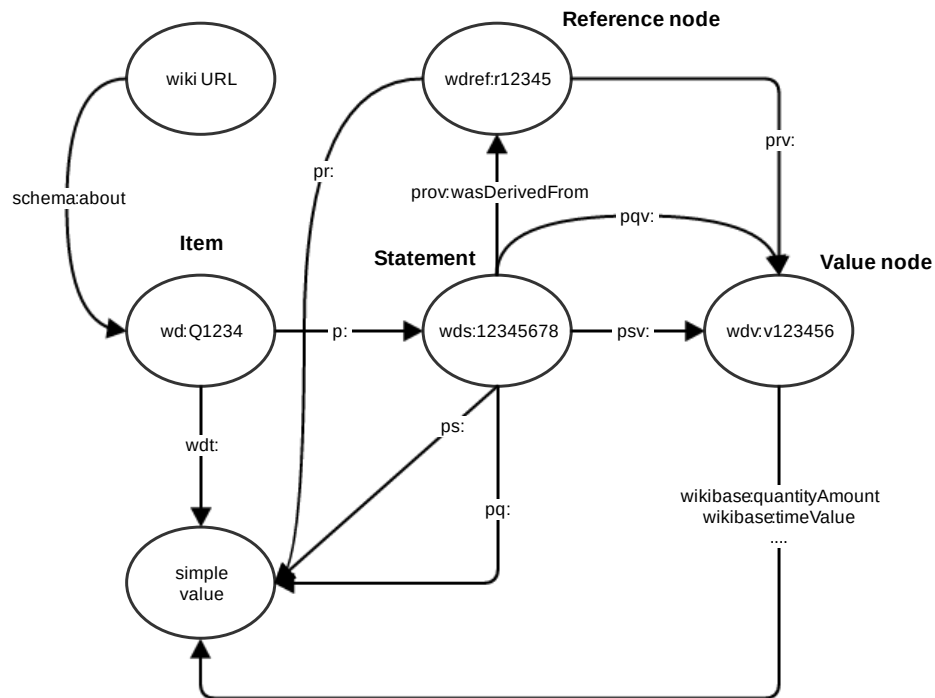


Abbildung 2.2: Mapping des Datenmodells auf RDF

Diese Übersetzung ist in Abb. 2.2 gezeigt. Jeder Pfeil beschreibt dabei Prädikatprefix, welches dann für RDF-Tripel verwendet wird. An dieses Präfix wird der Bezeichner der Property angehängen. Hier ist ein Beispiel zur Verdeutlichung (gekürzt, die Informationen zu den Reference und Value Nodes sind nicht abgebildet):

```

1  wd:Q42 p:P26 wds:q42-xxxx .
2  wds:q42-xxxx rdf:type wikibase:Statement .
3  wds:q42-xxxx ps:P26 wd:Q14623681 .
4  wds:q42-xxxx pq:P580 "1991-11-25T00:00:00Z"^^xsd:dateTime .
5  wds:q42-xxxx pqv:P580 wdv:c8ae0d38443d4671d3f893d7000a859e .
6  wds:q42-xxxx pq:P582 "2001-05-11T00:00:00Z"^^xsd:dateTime .
7  wds:q42-xxxx pqv:P582 wdv:1c30ade7914d072877b2db404a683d7c .
8  wds:q42-xxxx prov:wasDerivedFrom wds:yyyyyyyyy .
9  wd:Q42 wdt:P26 wd:Q14623681 .

```

Das Beispiel zeigt, wie die Property P26 (spouse) zuerst mit dem Prädikat p:P26 auf das Statement verlinkt. Das Statement verlinkt dann über ps:P26 zum einfachen Wert der Property. Mit pq:P580 (start time) bzw. pqv:P580 wird auf die einfache und komplexe Darstellung des Wertes für diesen Qualifier verlinkt. Die komplexe Darstellung enthält noch weitere Informationen zu dem Datum, wie bspw. die Genauigkeit oder das verwendete Kalendermodell.

Eine spezielle Bedeutung hat der `wdt` : Prefix. Tripel mit diesen Prädikaten verlinken Items direkt mit den einfachen Werten. Diese Tripel existieren allerdings nur dann, wenn das zugehörige Statement vom Typ `BestRank` hat. Dieser Typ umfasst alle Statements, die den höchsten Rang für diese Property haben und nicht deprecated sind. Solche Statements und Tripel werden auch als “truthy” bezeichnet. Die truthy Tripel sind sehr hilfreich, wenn man nicht an Qualifiern, Referenzen oder komplexeren Werten interessiert ist. Für diesen Fall bieten sie eine einfachere Sicht auf die Daten, was besonders bei Abfragen die Komplexität reduziert.

Anforderungen

3.1 funktionale Anforderungen

Im Fokus dieser Arbeit steht die Entwicklung eines Systems zur Filterung des RDF-Exports von Wikidata. Damit sollen Wissenschaftler mit wenig Aufwand Dumps nach speziellen Kriterien erstellen können. Die wesentlichen Merkmale des Systems sind:

Format Der Dump soll als RDF im N-Triples Format erstellt werden. Damit die gefilterten Dumps möglichst kompatibel mit dem vollständigen Wikidata RDF Dump sind, sollte das Schema dem offiziellen RDF-Dump-Format entsprechen. Erweiterungen des Schemas müssen klar gekennzeichnet sein.

Filterung Nutzer sollen über eine einfache Oberfläche wählen können, welche Entitäten in welchem Detailgrad im erstellten Dump enthalten sind.

Archivierung Damit die Dumps in wissenschaftlichen Veröffentlichungen zitiert werden können, muss die Verfügbarkeit auch in ferner Zukunft garantiert werden. Deshalb ist eine Methode zur Langzeitarchivierung generierter Dumps notwendig.

Suche es soll eine durchsuchbare Übersicht über alle erstellen Dumps. So können Ideen anderer Nutzer als Vorlage dienen und nicht jeder Nutzer muss sich eigene Filterregeln ausdenken,

Statistiken Um schnell zu entscheiden, ob ein Dump für einen bestimmten Anwendungsfall geeignet ist, sollten Statistiken über den Inhalt (z.B. die Anzahl der Entitäten) des Dumps angezeigt werden. Zusätzlich sollten auch schon während der Generierung Statistiken zum Fortschritt des Dumps bereitgestellt werden.

Nachvollziehbarkeit Gerade für den wissenschaftlichen Einsatz ist es erforderlich, dass die Herkunft der Daten und Generierung nachvollziehbar ist. Es sollte demnach leicht sichtbar sein, wie der Dump entstanden ist und eine Reproduktion des Dumps mit diesen Daten möglich sein.

Aktualität der Daten Die Dumps sollten aus möglichst aktuellen Daten erstellt werden. Es ist natürlich nicht notwendig, dass die Dumps immer komplett aktuell sind, aber die Daten zur Generierung der Dumps sollten regelmäßig aktualisiert werden.

3.2 nicht-funktionale Anforderungen

Neben den Anforderungen an die Funktion des Systems existieren auch eine Reihe von weiteren Anforderungen:

Hardwareanforderungen Der Ressourcenverbrauch des Systems sollte in einem akzeptablen Rahmen liegen. Als Anhaltspunkte für die Beurteilung dienen hier vergleichbare Systeme, wie bspw. der Wikidata Query Service, welcher ebenfalls Services basierend auf den RDF-Daten von Wikidata anbietet und gleichzeitig deutlich populärer ist. Demnach sollte das System entsprechend seiner Relevanz geringere Hardwareanforderungen als der Wikidata Query Service haben.

Freie Lizenz Die Wikimedia Foundation legt großen Wert darauf, möglichst viele der verwendeten Tools unter einer freien Lizenz bereitzustellen[@Pro13]. Wenn das System in der Wikimedia Cloud betrieben werden soll, dann ist die Veröffentlichung des Quellcodes unter einer freien Lizenz sogar Pflicht[@Pro19].

Bearbeitungszeit Das System sollte maximal einen Tag zur Generierung eines Dumps benötigen, besser unter 12 Stunden. Während längerer Prozesse sollte keine ständig Verfügbarkeit des Nutzers erwartet werden.

Erweiterbarkeit Da die funktionalen Anforderungen an das System sehr allgemein sind, muss auf Erweiterbarkeit geachtet werden sodass neue Anforderungen einfach umgesetzt werden können. Es sind beispielsweise viele verschiedene Filtermöglichkeiten und Statistiken denkbar, die nicht alle in der ersten Version umgesetzt werden können. Es ist daher sinnvoll vor allem in diesem Bereich auf Erweiterbarkeit zu achten.

Skalierbarkeit Wikidata wächst beständig, aktuell existieren etwas weniger als 59 Millionen Items. **[TODO cite]** Insgesamt gibt es über 700 Millionen Statements und diese Zahl ist allein im letzten Jahr um 200 Millionen gewachsen. Das System muss also mit dieser Menge an Daten umgehen können und dies auch für die nächsten Jahre noch leisten können.

3.3 Verwandte Arbeiten

Zu den verwandten Projekten gehört sicherlich der Wikidata Query Service[Mal+18], welcher bereits einen gezielten Zugriff auf die Daten gewährt. Die Daten des Query Services entsprechen bis auf wenige Minuten Verzögerung der neusten Wikidata Version. Da der Query Service allerdings beliebige SPARQL-Abfragen erlaubt, ist ein Timeout von einer Minute notwendig um zu großen Ressourcenverbrauch abzufangen, denn bei SPARQL-Abfragen kann die Termination nicht sichergestellt werden. Für die Erzeugung von großen Dumps ist der Query Service damit ungeeignet.

Zwei weitere existierende APIs, der Linked Data Export von Wikidata und die Wikidata API, skalieren ebenso nicht für größere Abfragen. Mit dem Linked Data Export können die RDF und JSON Daten einzelner Items über HTTP abgerufen werden. Die API erlaubt das Abrufen der JSON Daten von bis zu 50 Items in einer Anfrage. Beide Varianten sind daher nur für den Abruf einer kleinen Menge an Items geeignet.

Ein Mittelweg zwischen Abfragekomplexität für den Server und Anzahl an Abfragen an der Server beschreibt die Linked Data Fragments[Ver+16] Spezifikation. Allerdings führt dies zu hohen Abfragekosten auf dem Client, da die Joins auf dem Client ausgewertet werden. Die Implementierung für Wikidata basiert außerdem auf der selben RDF-Datenbank wie auch der Wikidata Query Service. Im Kontext von Wikidata haben Linked Data Fragments daher keinen Vorteil gegenüber dem Query Service.

Eine Variante zur Reduzierung der Größe von RDF-Dumps ist HDT[Fer+13], einem Binärformat für RDF-Daten. Es verwendet eine optimierte, komprimierte Kodierung zur Speicherung der Dumps. Auf diesem Format lassen sich sogar einfache SPARQL-Abfragen effizient ausführen[MAF12]. Leider erfordert die Erstellung von HDT-Dumps viel Arbeitsspeicher und Rechenzeit, besonders bei einer so großen Datenmenge wie Wikidata. Durch die komprimierte Kodierung ist eine inkrementelle Aktualisierung von HDT-Dumps nicht möglich, sodass der komplette Dump regelmäßig neu erstellt werden müsste.

DBpedia[Leh+15] teilt die seine Dumps in mehrere, kleinere Dumps auf. Auch einige aus Wikidata abgeleitete Dumps sind verfügbar¹. Durch das fest definierte Schema ist eine solche Trennung der Datensätze bei DBpedia möglich. Für Wikidata ist dieser Ansatz deutlich schwieriger, da das Datenschema so flexibel ist. Es gibt in Wikidata damit einzelnen Aspekt, nach dem diese Trennung erfolgen könnte.

¹<https://databus.dbpedia.org/dbpedia/wikidata/>

Ein mächtiges Tool zum Erstellen von Listen von Wikidata Items ist PetScan². Dieses Tool bietet ein User-Interface mit einer Vielzahl an Optionen, um Listen von Wikipedia bzw. Wikidata-Seiten zu erstellen. Wikidata ist nicht der Hauptanwendungszweck dieses Tools, dennoch ist es eine interessante Quelle von Filtern.

²<https://petscan.wmflabs.org/>

Design

In diesem Kapitel werden wir nun die Anforderungen präzisieren und ein Design für die Umsetzung erarbeiten. Dazu sind mehrere Aspekte relevant: zum einen die technische Umsetzung der Filterung und Generierung von Dumps (Architektur und unterstützte Filterkriterien), aber auch das User-Interface und welche externen Dienste genutzt werden können (z.B. für die Archivierung der erzeugten Dumps). Diese bedingen sich gegenseitig: es nützt wenig, wenn die Architektur sehr komplexe Filterkriterien unterstützt, dass User-Interface diese aber gar nicht abbilden kann (oder umgekehrt).

4.1 Architektur

Für die Architektur der Anwendung stellt sich zuerst die Frage, wie die Arbeit zwischen Server und Client verteilt werden soll. Auf der einen Seite steht die Variante, einen schlaun Client zu entwickeln welcher direkt mit der Wikidata API und dem SPARQL-Endpunkt spricht bzw. den Dump als Stream verarbeitet. Der Client kann dann von jedem Nutzer lokal auf seinem eigenen System ausgeführt werden. Der Vorteil dieser Variante ist klar: der Server müsste in diesem Fall keine eigenen Dumps erzeugen, sondern maximal Metadaten zu generierten Dumps speichern. Allerdings bringt diese Variante auch mehrere Probleme: erstens muss der Client vom Nutzer selbst ausgeführt werden, was problematisch wird wenn die Erzeugung der Dumps einen längeren Zeitraum in Anspruch nimmt, da dann das System des Nutzers die gesamte Zeit online sein muss. Zweitens müsste für die Archivierung der Nutzer die Dumps selbst hochladen. Die Generierung auf einem Server hat diese Nachteile nicht. Der Nutzer muss während der Generierung nicht online sein, sondern kann den Dump einfach herunterladen sobald dieser fertig ist. Außerdem kann die Archivierung vom Server übernommen werden. Die Anwendung implementiert die Logik zur Generierung der Dumps daher auf dem Server.

Die nächste Frage ist nun, wie der Server die Dumps generieren soll. Dafür sind drei Varianten denkbar:

- a) Der Server verwaltet einen eigenen Index
- b) Der Server nutzt den Wikidata SPARQL Endpunkt und die Wikidata API
- c) Der Server verarbeitet die existierend JSON oder RDF-Dumps von Wikidata auf Anfrage sequentiell

Variante a) könnte zum Beispiel so aussehen, dass die Daten vorher in kleine Blöcke zerlegt werden, von denen dann nur die gewünschten kombiniert werden. Insgesamt hat diese Variante den Vorteil, dass die Erzeugung der Dumps sehr schnell geht, denn es müssen nur die geforderten Blöcke gelesen werden. Allerdings ist die Variante relativ komplex zu implementieren, da der Index mit den aktuellen Daten von Wikidata synchron gehalten werden muss. Dazu ist entweder eine periodische Neuerstellung notwendig oder der Index muss inkrementell aktualisiert werden, was die Komplexität weiter erhöht. Die Notwendigkeit der inkrementellen Aktualisierung und gezielter Zugriff auf Teile der Daten hindert außerdem die Kompression. Diese Variante hat also einen hohen Speicherplatzbedarf.

Variante b) basiert auf der Beobachtung, dass Indexierung der Daten genau das ist, was auch RDF-Datenbanken machen. Es gibt schon eine existierende RDF-Datenbank für die Daten von Wikidata, die über den Wikidata Query Service abgefragt werden kann. Diese wird auch ständig aktuell gehalten, was das Problem der Aktualität löst. Die Generierung der Dumps in dieser Variante würde also aus den Filterkriterien einen SPARQL-Query generieren, und diesen dann an den SPARQL-Endpunkt von Wikidata senden. Wie auch Variante a) ist diese Möglichkeit sehr schnell, der Wikidata SPARQL Endpunkt besitzt ein Timeout von 60 Sekunden. Genau das ist aber auch ein Problem dieser Variante: das Timeout von 60 Sekunden erlaubt es nicht, sehr große Mengen von Daten über den Query Service anzufragen. Zum Beispiel terminiert diese Abfrage, welche alle Instanzen der Klasse Q13442814 (wissenschaftlicher Artikel) abfragt, nicht innerhalb des Timeouts (das Ergebnis wären 21986127 Entities):

```

1  SELECT ?item {
2    ?item wdt:P31 wd:Q13442814
3  }
```

Listing 4.1: SPARQL-Query nach wissenschaftlichen Artikeln

Und dieser Query fragt noch gar nicht nach Statements oder Labels der gefundenen Artikel. Dieses Problem ist auch schon in der Praxis vorgekommen[Pis+17b].

Die Variante c) hat dieses Problem nicht. Dafür ist hier die Generierung von Dumps deutlich langsamer, da für die Generierung ein vollständiger Wikidata-Export verarbeitet werden muss. Der Vorteil dieser Lösung ist, dass sie im Vergleich zu Variante

a) einfacher in der Umsetzung ist, da keine Synchronisation oder komplexe Indexstrukturen notwendig sind. Zudem ist kein zusätzlicher Speicherbedarf für einen Index erforderlich, der Wikidata-Export kann direkt als Datenstrom verarbeitet werden. Im Gegensatz zu Variante b) müssen die Filterkriterien nicht als SPARQL formulierbar sein, sondern können in der Programmiersprache implementiert werden, die für die Verarbeitung der Exports genutzt wird. Dafür sind komplexe, graph-basierte Filter die das Betrachten von mehr als einer Entität auf einmal erfordern, nicht so einfach umsetzbar.

Die Auswahl der Architektur ist auch davon abhängig, welche Kriterien zur Filterung unterstützt werden sollen. Ein paar Beispiele von Kriterien, nach denen andere Arbeiten die Exporte von Wissensgraphen wie Wikidata gefiltert haben, sind:

- 1) nur “truthy” Statements mit Entities als Objekt, Labels/Beschreibungen nur in Englisch und Spanisch[MH18]
- 2) nur truthy simple Statements, wobei sowohl Subjekt als auch Objekt ein Wikidata-Item sind[Nie17]
- 3) alle Items mit einem Statement für die Property P727 (Europeana ID, ID in der europäischen virtuellen Bibliothek „Europeana“ für Kulturobjekte)[FI19]
- 4) alle Items mit einem Statement für mindestens eine von 50 Properties[Met+19]
- 5) nur der englische Teil[Has+15]
- 6) nur Statements für Properties, wobei die Properties Instanzen einer bestimmten Klasse sein müssen, ohne deprecated, no-value und unknown-value Statements[HMS19] (allerdings wurde der Dump hier nicht als RDF, sondern in der JSON-Form verarbeitet)
- 7) nur Entities von gestorbenen Personen, die an mindestens drei Statements für mindestens 5000 mal vorkommende Properties beteiligt sind[Bor+11]
- 8) kleinere Datensätze, erzeugt durch zufälliges Auswählen von Statements[Mor+11]
- 9) alle Items, die Instanzen (P31) der Klassen Q5 (Mensch) sind und mindestens 6 Fakten (Statements) haben[HRC17] oder nur die einfachen Statements dieser Personen für die Properties P27 (Land der Staatsangehörigkeit) und P106 (Tätigkeit)[GMS15]

- 10) zufällige Auswahl von Entities (Sampling), mit englischer Beschreibung und mindestens 5 Statements[BM18], optional auch mit Downsampling von Instanzen bestimmter Klassen (um die Verteilung der Klassen im Ergebnis auszugleichen)[BM19]
- 11) alle Statements welche Referenzen haben, und die Sitelinks der Subjekte dieser Statements[Pis+17b]

Alle diese Filter sind für jede Entität lokal auswertbar, es müssen keine benachbarten Entitäten angeschaut werden. Nur die Entscheidung, für welche Properties die Statements exportiert werden sollen ist teilweise komplizierter. Bei einer sequentiellen Verarbeitung von Dumps ist die Zugehörigkeit einer Property zu einer Klasse erst bekannt, sobald die Entität für diese Property verarbeitet wurde. Da die Anzahl der Properties jedoch gering genug ist (< 10000), können solche komplexe Filter vor Beginn der Verarbeitung mithilfe des Wikidata Query Service in eine Liste von Properties aufgelöst werden. Die Variante c) kann damit die Filter dieser Art abdecken. Auch Sampling kann in dem sequentiellen Ansatz einfach implementiert werden.

Insgesamt bietet der sequentielle Ansatz die meiste Flexibilität und ist einfach in der Implementierung. Variante a) ist komplizierter und weniger flexibel, da ein Index nur bestimmte Filter effizient unterstützt. Variante b) funktioniert nur für kleinere Dumps, bei größeren Dumps das Problem des Timeouts auftritt. Die Einfachheit der Variante c) erlaubt es, schnell neue Arten von Filtern zu implementieren. In dieser Arbeit wurde sich deshalb für diese Variante entschieden, auch da zu erwarten ist dass das System am Anfang noch keine Nutzung in so großem Ausmaß haben wird, um die Komplexität und den zusätzlichen Speicherbedarf eines Index zu rechtfertigen.

4.2 Interface

Das Interface hat die Aufgabe, die Erstellung und Spezifikation von Dumps möglichst einfach zu gestalten und bereits existierende Dumps zu finden und anzuzeigen. Um das Interface für die Spezifikation der Filter zu entwerfen muss zunächst die Struktur der Filter, welche unterstützt werden sollen, genauer festgelegt werden.

Die Anwendungsfälle für das Filtern von Wikidata sind verschieden. Ein typisches Szenario ist die Extraktion von Daten einer bestimmten Thematik, z.B. nur Daten von Personen oder Städten. In diesem Fall geschieht die Filterung auf Entitätenebene, mit eventuell zusätzlichen Einschränkung der zu exportierenden Statements

und Sprachen. Ein anderes Szenario orientiert sich mehr an der Struktur der Statements. Beispiele hierfür sind die Analyse von lediglich Statements mit Referenzen, die Extraktion aller Statements für eine bestimmte Property oder alle Statements mit deprecated Rank. Das dritte Szenario verwendet Filterung, um die Daten für Benchmarks auf eine kleinere Menge zu reduzieren. In diesem Fall sind statistische Filter interessant, um z.B. nur jedes zehnte Statement oder nur jede zehnte Entität zu exportieren.

In Wikidata sind Statements immer ein Teil von Entities. Folglich kann der Filterprozess konzeptionell in zwei Schritte zerlegt werden: zuerst werden die Entities ausgewählt, danach für diese Entities die zu exportierenden Statements. Zusätzlich gibt es noch weitere globale Optionen, die sich nicht auf einzelne Statements beziehen, wie bspw. welche Sprachen exportiert werden sollen oder ob sitelinks von Entities exportiert werden sollen. Diese Struktur des Interfaces in Abb. 4.1 dargestellt.

Für die Auswahl der Entitäten sind viele verschiedene Arten von Bedingungen denkbar. Deswegen erlaubt das Interface die Kombination mehrerer Entitätsfilter von unterschiedlicher Art. Für die erste Version der Software ist zunächst eine Filterart geplant, die Auswahlkriterien auf Basis von vorhandenen Statements unterstützt. Mit diesem Filter kann gefordert werden, dass Statements für bestimmte Prädikate bzw. bestimmte Prädikat/Objekt-Paare für eine Entität existieren. Dieser Filter kann als Liste von Bedingungen im Interface dargestellt werden. Weitere Filter sind denkbar, wie zum Beispiel alle Entitäten die Ergebnis einer SPARQL-Abfrage sind. Prinzipiell könnte man Filter beliebig mit UND bzw. ODER kombinieren. Mit dieser Freiheit würde das Interface allerdings sehr komplex. Um das Interface einfach zu halten, werden Entitätsfilter deshalb immer mit ODER kombiniert, eine Entität wird demnach exportiert wenn mindestens einer der Filter zutrifft. Es ist auch möglich, keinen Filter anzugeben. In diesem Fall werden alle Entitäten exportiert (entsprechend den weiteren Regeln).

Für die Statements können Exportoptionen festgelegt werden. Dafür gibt zum einen Standardoptionen, die für alle Statements der selektierten Entitäten angewendet werden. Zusätzlich können für bestimmte Predikate spezielle Exportoptionen festgelegt werden. Damit können beispielsweise qualifier nur bei bestimmten Prädikaten exportiert werden. Die Optionen für den Export von Statements orientieren sich an der Struktur von Wikidata-Statements. Für jede Komponente eines Statements (simple Statement, full Statement, Qualifier und Referenzen) kann einzeln festgelegt werden, ob diese generiert werden soll oder nicht. Eine weitere Kategorie von Optionen betrifft den Export von Werten. Hier könnte man Schalter für den Export von einfachen Werten, komplexen Werten, normalisiert oder nicht normalisiert



Abbildung 4.1: Struktur des Interfaces zum Erstellen von Dumps

anbieten. Da das Backend dies aber aktuell noch nicht unterstützt sind diese Optionen aktuell im Interface noch nicht vorhanden. Insgesamt ist dieser Teil aber einfach erweiterbar, da die Optionen alle binär sind und somit einfach umsetzbar.

Die globalen Optionen sind auch größtenteils binär. Hier kann eingestellt werden, ob Labels, Beschreibungen, Aliase bzw. sitelinks für selektierte Entitäten exportiert werden sollen. Zudem findet sich hier die Option für die zu exportierenden Sprachen. Man kann entweder keine Filterung der Sprachen vornehmen (alle Sprachen exportieren) oder eine Liste von Sprachen angeben.

An allen Stellen wo dies sinnvoll ist bietet das Interface automatische Vervollständigung an. Das trifft auf alle Eingabefelder für Entitäten zu, wobei Entitäten hier über ihr Label vervollständigt werden. Außerdem wird eine Vervollständigung bei der Angabe von Sprache angeboten, wobei eine feste Liste der möglichen Sprachen verwendet wird.

4.3 Integration in existierende Infrastruktur

Um die Anwendung zu betreiben, werden Ressourcen in Form von Rechenzeit für die Generierung der Dumps und Speicherplatz für die Archivierung benötigt. Dazu lohnt es sich, bereits vorhandene Infrastruktur zu nutzen, um Aufwand und Kosten zu sparen.

Die Anwendung sollte auf Wikimedia-Servern betrieben werden, um die Verfügbarkeit auch in Zukunft unabhängig zu gewährleisten. Für das Verarbeiten der Dumps und Betreiben des Web-Interfaces bietet sich hier die Wikimedia Toolforge¹ an. Die Toolforge hat eine Grid-Engine, die zur Ausführung von längeren Jobs wie das Verarbeiten der Dumps geeignet ist, eine MySQL-Datenbank und einen Dienst zum Betreiben von Web-Services. Die Alternative zur Toolforge wäre ein Wikimedia Cloud VPS Projekt². Es wird jedoch empfohlen, Cloud VPS nur zu verwenden, falls die Toolforge nicht ausreichend ist. Deswegen wird für die Anwendung die Toolforge verwendet, da die Dienste in diesem Fall ausreichend sind.

Für die Archivierung der Dumps ist eine Integration mit Zenodo³ vorgesehen. Zenodo ist ein Datenarchivierungsdienst speziell für wissenschaftliche Zwecke und besitzt auch die Möglichkeit, einen Digital Object Identifier (DOI) zu registrieren, wo-

¹<https://tools.wmflabs.org/>

²https://wikitech.wikimedia.org/wiki/Portal:Cloud_VPS

³<https://zenodo.org/>

mit die Referenzierung von generierten Datensätzen leicht möglich ist. Damit wird die Anforderung an Langzeitarchivierung erfüllt.

Die Integration mit Zenodo funktioniert so, dass abgeschlossene Dumps als Datensatz zu Zenodo hochgeladen werden. Dabei stellt sich die Frage, welcher Account für den Upload verwendet wird. Zenodo bietet auch einen OAuth-Schnittstelle an, sodass es möglich wäre, die Dumps direkt zu einem Account des Nutzers hochzuladen. Das erfordert es allerdings, dass Nutzer bereits einen Zenodo-Account besitzen. Da die Implementierung von OAuth zudem zusätzliche Komplexität verursacht, verwendet die erste Version der Anwendung einen eigenen Account, der speziell dafür erstellt wurde. Damit ist es mit einem Klick möglich, einen Dump zu Zenodo hochzuladen und den Dump dann über die DOI zu referenzieren, ohne das Nutzer selbst einen Account bei Zenodo benötigen.

Implementierung

Die Anwendung besteht aus zwei Teilen: Backend und Frontend. Über das Frontend können Nutzer Aufträge für Dumps erstellen und existierende Dumps verwalten, während das Backend nur für die Generierung von Dumps zuständig ist. Das Backend stellt dazu in regelmäßigen Abständen Anfragen an eine Datenbank, um nach neuen Aufträgen zu schauen welche vom Frontend hinzugefügt wurden. Man könnte dafür auch eine spezielle Message-Queue verwenden. Die Datenbank wäre in diesem Fall allerdings trotzdem notwendig, um Metadaten zu den Aufträgen persistent zu speichern. Um die Komplexität gering zu halten wurde deshalb auf eine separate Message-Queue verzichtet.

In diesen Kapitel werden wir uns zunächst das Datenmodell anschauen, welches zur Kommunikation verwendet wird. Danach werden wir die Implementierung von jeweils Frontend und Backend genauer betrachten.

5.1 Datenmodell

Eine Übersicht des Datenmodells liefert Abb. 5.1. Das zentrale Element ist der Dump, welcher alle Metadaten zu einem Auftrag speichert. Neben ein paar einfachen Daten wie Titel (`title`) und Zeitpunkt der Erstellung des Auftrags (`created_at`) bzw. Fertigstellung (`finished_at`) ist jeder Dump durch eine JSON-Spezifikation (`spec`) charakterisiert. Zusätzlich hat der Dump auch Felder für Statistiken, wie die Anzahl der Entitäten (`entity_count`) und Dateigröße (`compressed_size`).

Für jeden Durchlauf des Backends wird ein Run angelegt. Die von diesem Durchlauf verarbeiteten Aufträge verweisen dann auf den Run. Damit der aktuelle Fortschritt ermittelt werden kann, wird die Anzahl der bereits verarbeiteten Entitäten in dem Attribut `count` gespeichert. Da die Anzahl von Entitäten in dem vollem Wikidata-Dump bekannt ist, lässt sich daraus der Fortschritt errechnen.

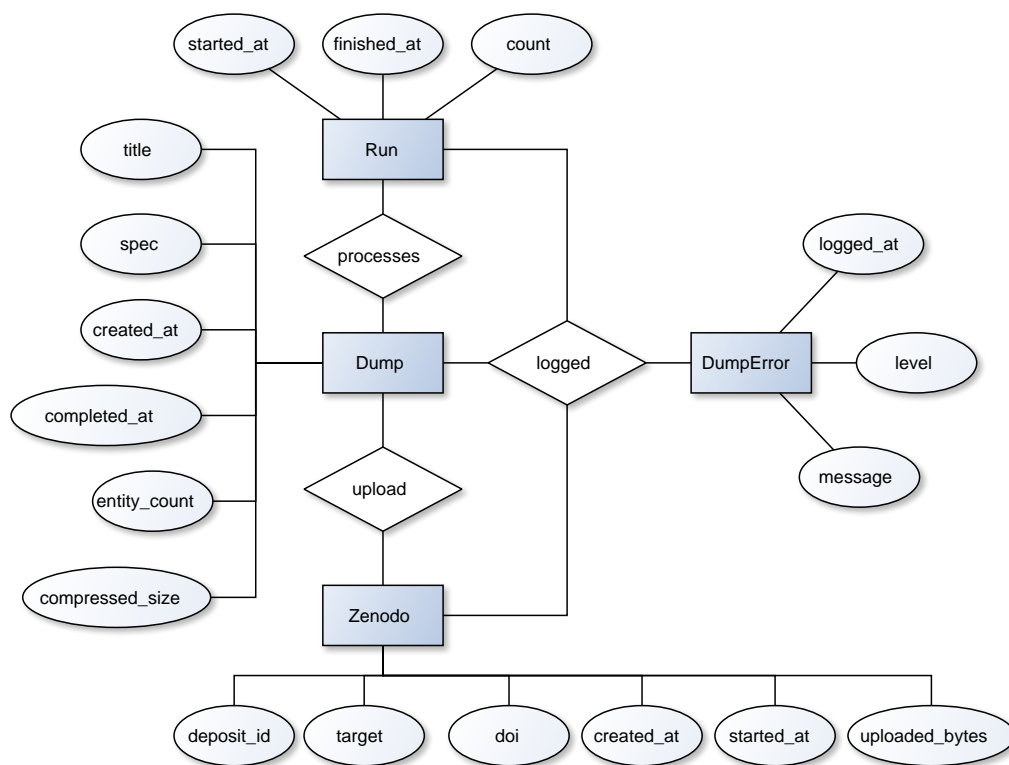


Abbildung 5.1: Datenbankschema

5.2 Backend

Für die Implementierung des Backends wird das Wikidata-Toolkit verwendet. Da Wikidata-Toolkit in Java geschrieben ist, muss deswegen auch eine Java-kompatible Programmiersprache verwendet werden. An dieser Stelle wurde Java gewählt, um auch die Wartbarkeit der Anwendung in Zukunft sicherzustellen, da Java im Vergleich zu anderen Programmiersprachen mit Java-Kompatibilität (wie zum Beispiel Scala¹ oder Kotlin²) deutlich weiter verbreitet und bekannter ist.

Die Hauptschleife des Backends besteht aus zwei Phasen, Warten und Verarbeitung. Während der wartenden Phase wird kontinuierlich auf neue Aufträge gewartet. Sobald neue Aufträge verfügbar sind, wird die Verarbeitung gestartet. Neue Aufträge werden dann erst wieder nach Beendigung des aktuellen Verarbeitungsprozesses abgerufen.

Am Start befindet sich das Backend in der wartenden Phase. Um neue Aufträge abzurufen, werden dazu periodische die in Listing 5.1 dargestellten SQL-Befehl in einer Transaktion ausgeführt. Da in Zeile 6 nur Aufträge zugewiesen werden, die noch keinem Run zugewiesen sind, kann diese Abfrage theoretisch auch von mehreren Prozessen gleichzeitig ausgeführt werden ohne dabei Kollisionen zu erzeugen. Es ist so unmöglich, dass ein Auftrag mehr als einem Run zugewiesen wird, was die Robustheit des Systems erhöht. Falls diese Befehle keine Ergebnisse liefern (wenn keine neuen Aufträge vorliegen), wird eine definierte Zeit gewartet bevor dieser Vorgang wiederholt wird. Diese Wartezeit führt gleichzeitig dazu, dass mehrere Aufträge gesammelt werden können, da die Verarbeitung nicht direkt beginnt.

```
1  -- neuen Run erstellen
2  INSERT INTO run () VALUES ()
3  -- generated id: 1
4
5  -- Aufträge dem Run zuweisen
6  UPDATE dump SET run_id = 1 WHERE run_id IS NULL
7
8  -- Zugewiesene Aufträge abrufen
9  SELECT id, spec FROM dump WHERE run_id = :run
```

Listing 5.1: Abrufen neuer Aufträge

Zum Verarbeiten der Aufträge wurde der in Wikidata-Toolkit bereits vorhandene RDF-Export angepasst. Der RDF-Export ist dabei als eine Klasse implementiert, wel-

¹<https://www.scala-lang.org/>

²<https://kotlinlang.org/>

che das von Wikidata-Toolkit erwartete Interface `EntityDocumentProcessor` implementiert (Listing 5.2).

```
1 public interface EntityDocumentProcessor {
2     void processItemDocument(ItemDocument itemDocument);
3     void processPropertyDocument(PropertyDocument propertyDocument);
4     void processLexemeDocument(LexemeDocument lexemeDocument);
5 }
```

Listing 5.2: `EntityDocumentProcessor` Interface

Listing 5.3 zeigt den Ablauf des Exports in Pseudocode. Für jede Entität (Items, Properties und Lexemes) wird dazu zunächst überprüft ob sie überhaupt exportiert werden soll. Nur wenn das der Fall ist, werden danach für jedes Statement die Optionen zum Export entsprechend der Filter-Spezifikation bestimmt. Wenn die Optionen feststehen, kann dann das Statement exportiert werden. Aktuell werden Lexemes noch nicht unterstützt, da Wikidata-Toolkit den RDF-Export dafür noch nicht implementiert hat. Wenn ein Lexeme exportiert werden soll wird deshalb ein Fehler erzeugt. Zusätzlich zu den Statements wird noch RDF für Labels, Descriptions, Aliases, Sitelinks und Metadaten zu der Entität erzeugt, falls entsprechend der Filter-Spezifikation verlangt.

```
1 for each entity:
2     if spec includes entity:
3         if entity is lexeme: raise error
4
5         if spec.labels: export entity labels
6         if spec.aliases: export entity aliases
7         if spec.descriptions: export entity descriptions
8
9         for each statement:
10             let options = get options for statement from spec
11             export statement with options
12
13         if spec.sitelinks: export entity sitelinks
14         if spec.meta: export entity metadata
```

Listing 5.3: Export Pseudocode

Wikidata-Toolkit unterstützt mehrere `EntityDocumentProcessors` gleichzeitig. Damit können mehrere Aufträge in einem Durchlauf verarbeitet werden. Diese Funktionalität wird auch verwendet, um den aktuellen Fortschritt des Durchlaufs in der Datenbank zu aktualisieren. Dazu zählt ein `EntityDocumentProcessor` die Anzahl der verarbeiteten Entities mit und speichert diese regelmäßig in der `run`-Tabelle der Datenbank.

5.3 Frontend

Das Frontend besteht aus einem Web-Interface zum Erstellen von Dump-Aufträgen und Verwaltung der existierenden Dumps. Für dessen Implementierung wird hauptsächlich HTML/CSS mit Typescript verwendet, für die Auslieferung und Kommunikation mit der Datenbank gibt es aber auch eine kleine serverseitige Anwendung, welche in Python mit Flask geschrieben ist.

Der serverseitige Teil bietet dazu Endpunkte für das Erstellen, Suchen, Herunterladen und Abfrage von Informationen von Dumps an. Dazu werden vier Endpunkte bereitgestellt:

- `POST /create` erstellt einen neuen Dump-Auftrag. Der Request-Body werden die Filter-Spezifikation sowie ein paar Metadaten (Titel, etc.) übergeben.
- `GET /dump/<id>` liefert eine Statusseite mit Informationen zu einem bestimmten Dump.
- `GET /dumps` gibt eine Liste aller Dumps zurück.
- `GET /download/<id>` lädt einen Dump herunter.

Evaluation

Zur Evaluation der Anwendung wird diesem Kapitel bewertet, ob Wikidata-Toolkit eine geeignete Wahl zum Erzeugen der RDF-Dumps ist. Dazu wird das von Wikidata-Toolkit erzeugte RDF mit den offiziellen Wikidata RDF-Exporten verglichen, um die Korrektheit und Vollständigkeit zu überprüfen.

6.1 Vorgehen

Für den Vergleich werden die JSON- und RDF-Exporte von Wikidata vom 26. August 2019 verwendet [TODO cite]. Da sich die Daten während der Generierung des Exports verändern, beschreiben die JSON- und RDF-Exporte nicht exakt die selben Daten. Deswegen werden für den Vergleich zusätzlich noch die Wikidata incremental Dumps verwendet. Diese enthalten die JSON-Version aller in einem Zeitraum geänderten Wikidata Dokumente, zusammen mit Revisionsnummer und Datum der Revision. Damit können Dokumente des JSON-Dumps, deren Revisionsnummer nicht identisch zu dem entsprechenden Dokument in dem RDF-Export ist, durch die aktualisierte Version ersetzt werden. Um die Revisionsnummer eines Dokuments des RDF-Exports zu bestimmen wird das mit dem Prädikat `schema:dateModified` angegebene Änderungsdatum verwendet.

Nachdem beide Exporte so auf die selbe Version gebracht wurden, wird mit der Anwendung aus dem JSON-Export RDF erzeugt. Dazu werden keine Filter verwendet, entsprechend sollte die Anwendung in diesem Fall den RDF-Export komplett reproduzieren. Praktisch ist dabei, dass die Dokumente sowohl in dem JSON-Export als auch in dem RDF-Export in der selben Reihenfolge gespeichert sind. Damit kann der Vergleich für jedes Paar von Dokumenten separat erfolgen.

Ein Problem für den Vergleich stellen die Value Nodes dar. Die IRI für Value Nodes enthält im RDF-Export von Wikidata einen Hash, der auf dem internen Datenmodell basiert und daher schwer zu reproduzieren ist. Aus diesem Grund stimmen die von Wikidata-Toolkit generierten Tripel nicht eins-zu-eins mit denen des RDF-Exports

überein, da die IRIs von Values Nodes einen anderen Hash benutzen. Um dieses Problem zu lösen, muss zur Bestimmung der Differenzen eine Abbildung der IRIs aus dem Export zu den IRIs der generierten Daten bestimmt werden. Das ist allerdings nicht sonderlich schwer, da Value Nodes immer nur von Nodes mit festen IRIs referenziert werden (Referenz und Statement Nodes). Die Value Nodes können deswegen über ihre Verbindungen zu diesen Knoten identifiziert und abgeglichen werden. Diese Lösung funktioniert auch zum Abgleich der im Export vorkommenden Blank Nodes.

Nach diesem Abgleich werden die Tripel verglichen. Die Differenzen sind die Tripel, die entweder nur im Export oder nur in dem generierten RDF vorkommen. Einige Differenzen sind schon vorher bekannt, die bspw. aufgrund von fehlenden Funktionen in Wikidata-Toolkit entstehen. Zur Vereinfachung der Analyse werden folgende Tripel von dem Vergleich ausgenommen:

1. Tripel, deren Subjekt das Prefix `wdata:` besitzt. Diese Tripel gehören zu den Data Nodes, welche aktuell von Wikidata-Toolkit nicht generiert werden. Die Data Nodes enthalten Metainformation zu dem Entity, wie `schema:dateModified`.
2. Einige Entites sind nur im RDF-Export oder nur im JSON-Export enthalten. Das liegt zum einen an Löschungen in der Zeit zwischen der Erstellung beider Exports. Außerdem enthalten die RDF-Exporte auch Entities, die mittlerweile eine Weiterleitung auf ein anderes Entity sind. Diese Entities generiert Wikidata-Toolkit aber noch nicht. Diese Entities werden daher ignoriert.
3. Tripel, die normalisierte Werte für Properties angeben und die entsprechenden Value Nodes. Wikidata-Toolkit unterstützt normalisierte Werte noch nicht, weshalb ein Vergleich hier nicht möglich ist.
4. Tripel mit Prädikat `schema:name` oder `skos:prefLabel`. Diese Tripel sind Duplikate der Tripel mit Prädikat `rdfs:label`, und Wikidata-Toolkit generiert aktuell nur `rdfs:label`-Tripel.
5. Triple mit den Prädikaten `owl:onProperty` oder `owl:complementOf` sowie Tripel mit einem der Objekte `owl:Class`, `owl:Thing`, `owl:Restriction`, `owl:DatatypeProperty` oder `owl:ObjectProperty`. Diese Tripel werden für OWL-Definitionen der Wikidata Properties verwendet. Im RDF-Export finden sich diese Tripel in dem Dokument für die Property, Wikidata-Toolkit generiert die Tripel jedoch wenn die Property das erste mal verwendet wird. Damit lassen sich diese Tripel nicht dokumentweise vergleichen.

6.2 Ergebnis

Eine Übersicht über das Ergebnis des Vergleichs gibt Tabelle 6.1. Die meisten Differenzen lassen sich einfach beheben, oft sind nur wenige Veränderungen an Wikidata-Toolkit notwendig. Dazu zählen: Statement ID, Sitelinks Siteinfo, invalide IRIs, simple value (Referenz), complex value (SomeValue), Koordinaten, Dezimalzahlen, Commons Link, Formeln, falsche Literaltypen, Datum deduplizierung und Referenz deduplizierung. Diese Korrekturen werden auch dem Wikidata-Toolkit Projekt beigetragen. Ein Großteil der Differenzen treten bei komplexen Literalwerten, wie Zeitdaten oder Geokoordinaten auf. Während die meisten Differenzen relativ harmlose Formfehler sind (wie das Verwenden falscher Literaltypen), sind auch drei inhaltliche Fehler aufgefallen.

Der erste Fall ist die Vertauschung von Längen- und Breitengrad bei Koordinatenangaben. Statt `"Point(4.6681 50.6411)"^^geo:wktLiteral` generiert Wikidata-Toolkit `"Point(50.641111111111 4.6680555555556)"^^geo:wktLiteral` (die Abweichung der Fließkommazahlen liegen an dem Rundungsproblem). Die GeoSPARQL Spezifikation[12] legt für den Datentyp `geo:wktLiteral` fest, dass die Reihenfolge der Koordinaten abhängig vom angegebenen Koordinatensystem ist. Für den Standardwert CRS84¹ ist die Reihenfolge *Längengrad-Breitengrad*, was entgegen der verbreiteten Darstellung 50°38'28N, 4°40'5"E ist.

Der zweite Fall hängt mit der Deduplizierung von Value Nodes zusammen. Um die Größe des Dumps zu reduzieren, werden für gleiche komplexe Werte die selben Value Nodes verwendet, sodass die Daten nicht zweimal abgespeichert werden müssen. Wikidata-Toolkit verwendet dazu einfach den Hash, der auch in der IRI für die Value Node vorkommt. Falls schon ein Value Node für diesen Hash generiert wurde, werden die Daten nicht erneut geschrieben. Bei Daten und Zeitwerten berücksichtigt dieser Hash jedoch die Präzision des Wertes nicht. Damit werden Werte, die sich nur in der Präzisionsangabe unterscheiden, zum selben Value Node. Die Präzision aller Zeitwerte für diesen Zeitpunkt entspricht dann der Präzision des zuerst verarbeiteten Zeitwerts. Dieser Fehler lässt sich einfach beheben, in dem auch die Präzision mit in den Hash aufgenommen wird. Ein ähnliches Problem tritt im Zusammenhang mit Referenzen auf. Hier wird auch die Reihenfolge der Properties vom Hash berücksichtigt, sodass Referenzen mit gleichem Inhalt aber unterschiedlicher Reihenfolge unterschiedliche Hashes erhalten. Diese Differenz ist natürlich deutlich weniger kritisch (ob diese Differenz überhaupt einen inhaltlichen Fehler darstellt ist streitbar) lässt sich aber genauso einfach beheben.

¹<http://www.opengis.net/def/crs/OGC/1.3/CRS84>

Stichwort	Beschreibung
wdata:	Nicht implementiert
normalisierte Werte	Nicht implementiert
owl:sameAs	Weiterleitungen nicht implementiert
Statement ID	Groß/Kleinschreibung der Item-ID in der Statement ID nicht beachtet
Language Tags	Unterschiedliche Language Tags
simple value (Referenz)	Bei Referenzen werden nur komplexe Werte angegeben
complex value (SomeValue)	SomeValue Snaks wird auch ein BNode für den komplexen Wert generiert
Sitelinks Siteinfo	bei Sitelinks fehlen Tripel für schema:name, schema:isPartOf, wikibase:wikiGroup
invalide IRIs	URL-Werte mit " erzeugen invalides RDF
Koordinaten	Längen- und Breitengrad in Koordinatenwerten vertauscht
Dezimalzahlen	Nicht konforme Verwendung von wissenschaftlicher Darstellung
Kalender	Umrechnung von Datumsangaben in gregorianischen Kalender fehlt
invalide Daten	Mehrere Abweichungen bei invaliden Daten (30 Februar etc)
Commons Link	Links zu Wikimedia Commons sind unterschiedlich (falsches Prefix und Behandlung von Sonderzeichen)
Formeln	Konvertierung nach MathML fehlt
falsche Literaltypen	xsd:int vs xsd:integer, Tippfehler bei wikibase:GlobeCoordinatesValue, fehlender Datentyp wikibase:MusicalNotation
Rundung	Gleitkommazahlen unterscheiden sich aufgrund von Rundungsfehlern
Datumeduplizierung	Datumsangaben wurden falsch dedupliziert (Präzision nicht beachtet)
Referenzdeduplizierung	Gleiche Referenzen mit unterschiedlicher Snak-Reihenfolge werden nicht dedupliziert

Tabelle 6.1: Übersicht der festgestellten Differenzen (aus Sicht von Wikidata-Toolkit)

Der dritte Fall betrifft auch Datumsangaben. Die XML-Schema Spezifikation verlangt, dass Literale mit Typ `xsd:dateTime` im proleptischen gregorianischen Kalender angegeben werden.² Im JSON-Export sind jedoch auch Daten im julianischen Kalender vorhanden. Wikidata-Toolkit hat die notwendige Konversion zum gregorianischen Kalender noch nicht implementiert. Dieser Fehler kann die Daten um mehrere Tage verändern. Ein Beispiel dafür sind diesen beiden Tripel:

```
1 # tripel aus dem Wikidata RDF Export
2 wd:Q45 wdt:P571 "1143-10-12T00:00:00Z"^^xsd:dateTime .
3
4 # tripel von Wikidata-Toolkit generiert
5 wd:Q45 wdt:P571 "1143-10-05T00:00:00Z"^^xsd:dateTime .
```

Datumsangaben sind allgemein komplex, was sich auch in weiteren Differenzen zeigt. Diese sind zwar inhaltlich nicht falsch, entsprechen aber trotzdem nicht der Spezifikation. Ein Beispiel sind negative Jahreszahlen. Die XML Schema Spezifikation verlangt hier, dass die Jahreszahl (ohne das Minuszeichen) vierstellig ist. Wikidata-Toolkit generiert aber insgesamt nur vier Stellen (mit Minuszeichen). Weiterhin gibt es Abweichungen bei invaliden Daten. Der RDF-Export von Wikidata versucht diese zu bereinigen. Wenn Tag oder Monat 0 ist, wird dieser auf 1 geändert und der Tag wird auf die maximale Anzahl an Tagen für diesen Monat begrenzt. Das wird auch von der XML-Schema Spezifikation so verlangt. Wikidata-Toolkit verändert die Daten jedoch deutlich weniger, sodass sich die Tripel hier unterscheiden.

Weitere Differenzen finden sich bei den Language Tags. Das Problem tritt hier bei weniger verbreiteten Sprachen auf, die teilweise keinen Language Code nach BCP47³ besitzen. Wikidata Toolkit versucht hier, die Wikimedia Language Codes auf BCP47 Codes abzubilden. Diese Abbildung wird im RDF Export von Wikidata nicht vorgenommen. In diesem Fall ist es schwer zu sagen, welche Variante "korrekt" ist, da RDF explizit die Verwendung von BCP47 vorsieht. Dennoch sollte sich am Ende für eine einheitliche Variante entschieden werden. Die Verwendung der Wikimedia Language Codes, die sich auch an BCP47 orientieren, stellt hier die einfachste Lösung da.

Insgesamt zeigt dieser Vergleich, dass das komplexe Datenmodell von Wikidata auch einige Tücken hat. Viele der gefundenen Differenzen sind subtil und wären ohne explizites Suchen danach vermutlich schwer zu finden. Durch die Verbesserung von Wikidata-Toolkit können diese Differenzen reduziert werden.

²<https://www.w3.org/TR/xmlschema11-2/#dateTime>

³<https://tools.ietf.org/html/bcp47>

Fazit

7

Literatur

- [BM19] Rajarshi Bhowmik und Gerard de Melo. “Be Concise and Precise: Synthesizing Open-Domain Entity Descriptions from Facts”. In: *CoRR* abs/1904.07391 (2019). arXiv: 1904.07391 (siehe S. 18).
- [BM18] Rajarshi Bhowmik und Gerard de Melo. “Generating Fine-Grained Open Vocabulary Entity Type Descriptions”. In: *CoRR* abs/1805.10564 (2018). arXiv: 1805.10564 (siehe S. 18).
- [Bor+11] Antoine Bordes, Jason Weston, Ronan Collobert und Yoshua Bengio. “Learning Structured Embeddings of Knowledge Bases”. In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. Hrsg. von Wolfram Burgard und Dan Roth. AAAI Press, 2011 (siehe S. 17).
- [Bra+16] Freddy Brasileiro, João Paulo A. Almeida, Victorio Albani de Carvalho und Giancarlo Guizzardi. “Applying a Multi-Level Modeling Theory to Assess Taxonomic Hierarchies in Wikidata”. In: *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11-15, 2016, Companion Volume*. Hrsg. von Jacqueline Bourdeau, Jim Hendler, Roger Nkambou, Ian Horrocks und Ben Y. Zhao. ACM, 2016, S. 975–980 (siehe S. 1).
- [Erx+14] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez und Denny Vrandečić. “Introducing Wikidata to the Linked Data Web”. In: *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*. Hrsg. von Peter Mika, Tania Tudorache, Abraham Bernstein u. a. Bd. 8796. Lecture Notes in Computer Science. Springer, 2014, S. 50–65 (siehe S. 8).
- [Fer+13] Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez, Axel Polleres und Mario Arias. “Binary RDF representation for publication and exchange (HDT)”. In: *J. Web Semant.* 19 (2013), S. 22–41 (siehe S. 13).
- [FI19] Nuno Freire und Antoine Isaac. “Technical usability of Wikidata’s linked data Evaluation of machine interoperability and data interpretability”. In: 2019 (siehe S. 17).
- [GMS15] Doron Goldfarb, Dieter Merkl und Maximilian Schich. “Quantifying Cultural Histories via Person Networks in Wikipedia”. In: *CoRR* abs/1506.06580 (2015). arXiv: 1506.06580 (siehe S. 17).

- [HRC17] Ben Hachey, Will Radford und Andrew Chisholm. “Learning to generate one-sentence biographies from Wikidata”. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*. Hrsg. von Mirella Lapata, Phil Blunsom und Alexander Koller. Association for Computational Linguistics, 2017, S. 633–642 (siehe S. 17).
- [HMS19] Tom Hanika, Maximilian Marx und Gerd Stumme. “Discovering Implicational Knowledge in Wikidata”. In: *CoRR abs/1902.00916* (2019). arXiv: 1902.00916 (siehe S. 17).
- [Has+15] Oktie Hassanzadeh, Michael J. Ward, Mariano Rodriguez-Muro und Kavitha Srinivas. “Understanding a large corpus of web tables through matching with knowledge bases: an empirical study”. In: *Proceedings of the 10th International Workshop on Ontology Matching collocated with the 14th International Semantic Web Conference (ISWC 2015), Bethlehem, PA, USA, October 12, 2015*. Hrsg. von Pavel Shvaiko, Jérôme Euzenat, Ernesto Jiménez-Ruiz, Michelle Cheatham und Oktie Hassanzadeh. Bd. 1545. CEUR Workshop Proceedings. CEUR-WS.org, 2015, S. 25–34 (siehe S. 17).
- [Leh+15] Jens Lehmann, Robert Isele, Max Jakob u. a. “DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia”. In: *Semantic Web 6.2* (2015), S. 167–195 (siehe S. 13).
- [Mal+18] Stanislav Malyshev, Markus Krötzsch, Larry González, Julius Gonsior und Adrian Bielefeldt. “Getting the Most Out of Wikidata: Semantic Technology Usage in Wikipedia’s Knowledge Graph”. In: *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part II*. Hrsg. von Denny Vrandečić, Kalina Bontcheva, Mari Carmen Suárez-Figueroa u. a. Bd. 11137. Lecture Notes in Computer Science. Springer, 2018, S. 376–394 (siehe S. 13).
- [MAF12] Miguel A. Martínez-Prieto, Mario Arias Gallego und Javier D. Fernández. “Exchange and Consumption of Huge RDF Data”. In: *The Semantic Web: Research and Applications*. Hrsg. von Elena Simperl, Philipp Cimiano, Axel Polleres, Oscar Corcho und Valentina Presutti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, S. 437–452 (siehe S. 13).
- [Met+19] Daniele Metilli, Valentina Bartalesi, Carlo Meghini und Nicola Aloia. “Populating Narratives Using Wikidata Events: An Initial Experiment”. In: *Digital Libraries: Supporting Open Science - 15th Italian Research Conference on Digital Libraries, IRCDL 2019, Pisa, Italy, January 31 - February 1, 2019, Proceedings*. Hrsg. von Paolo Manghi, Leonardo Candela und Gianmaria Silvello. Bd. 988. Communications in Computer and Information Science. Springer, 2019, S. 159–166 (siehe S. 17).

- [MH18] José Moreno-Vega und Aidan Hogan. “GraFa: Faceted Search & Browsing for the Wikidata Knowledge Graph”. In: *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 8th - to - 12th, 2018*. Hrsg. von Marieke van Erp, Medha Atre, Vanessa López, Kavitha Srinivas und Carolina Fortuna. Bd. 2180. CEUR Workshop Proceedings. CEUR-WS.org, 2018 (siehe S. 17).
- [Mor+11] Mohamed Morsey, Jens Lehmann, Sören Auer und Axel-Cyrille Ngonga Ngomo. “DBpedia SPARQL Benchmark - Performance Assessment with Real Queries on Real Data”. In: *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*. Hrsg. von Lora Aroyo, Chris Welty, Harith Alani u. a. Bd. 7031. Lecture Notes in Computer Science. Springer, 2011, S. 454–469 (siehe S. 17).
- [Nie17] Finn Årup Nielsen. “Wembedder: Wikidata entity embedding web service”. In: *CoRR abs/1710.04099* (2017). arXiv: 1710.04099 (siehe S. 17).
- [12] *OGC GeoSPARQL - A Geographic Query Language for RDF Data*. OGC 11-052r4. Open Geospatial Consortium. 2012 (siehe S. 30).
- [Pis+17a] Alessandro Piscopo, Lucie-Aimée Kaffee, Chris Phethean und Elena Simperl. “Provenance Information in a Collaborative Knowledge Graph: An Evaluation of Wikidata External References”. In: *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*. Hrsg. von Claudia d’Amato, Miriam Fernández, Valentina A. M. Tamma u. a. Bd. 10587. Lecture Notes in Computer Science. Springer, 2017, S. 542–558 (siehe S. 1).
- [Pis+17b] Alessandro Piscopo, Pavlos Vougiouklis, Lucie-Aimée Kaffee u. a. “What do Wikidata and Wikipedia Have in Common?: An Analysis of their Use of External References”. In: *Proceedings of the 13th International Symposium on Open Collaboration, OpenSym 2017, Galway, Ireland, August 23-25, 2017*. Hrsg. von Lorraine Morgan. ACM, 2017, 1:1–1:10 (siehe S. 1, 2, 16, 18).
- [Ver+16] Ruben Verborgh, Miel Vander Sande, Olaf Hartig u. a. “Triple Pattern Fragments: a Low-cost Knowledge Graph Interface for the Web”. In: *Journal of Web Semantics* 37–38 (März 2016), S. 184–206 (siehe S. 13).
- [VK14] Denny Vrandečić und Markus Krötzsch. “Wikidata: A Free Collaborative Knowledge Base”. In: *Communications of the ACM* 57 (2014), S. 78–85 (siehe S. 1).

Webpages

- [@Gri18] Hillary K. Grigonis. *Wikibase/Indexing/RDF Dump Format*. 2018. URL: <https://web.archive.org/web/20190822103829/https://www.digitaltrends.com/social-media/facebook-about-this-article-wikipedia/> (besucht am 22. Aug. 2019) (siehe S. 1).

- [@Pro19] Wikimedia Project. *Wikitech:Cloud Services Terms of use*. 2019. URL: https://web.archive.org/web/20190804123117/https://wikitech.wikimedia.org/wiki/Wikitech:Cloud_Services_Terms_of_use (besucht am 4. Aug. 2019) (siehe S. 12).
- [@Pro13] Wikimedia Foundation Project. *Resolution: Wikimedia Foundation Guiding Principles*. 2013. URL: https://web.archive.org/web/20190804122555/https://foundation.wikimedia.org/wiki/Resolution:Wikimedia_Foundation_Guiding_Principles (besucht am 4. Aug. 2019) (siehe S. 12).

