Generierung angepasster RDF-Dumps von Wikidata

Benno Fünfstück

November 4, 2019
Version: Final

Technische Universität Dresden

Fakultät Informatik
Institut für Theoretische Informatik
Professur für Wissensbasierte Systeme

Bachelorarbeit

Generierung angepasster RDF-Dumps von Wikidata

Benno Fünfstück

1. Gutachter Prof. Dr. Markus Krötzsch
Fakultät Informatik
Technische Universität Dresden

2. Gutachter Prof. Sebastian Rudolph
Fakultät Informatik
Technische Universität Dresden

Betreuer Prof. Dr. Markus Krötzsch

November 4, 2019

Benno Fünfstück
Generierung angepasster RDF-Dumps von Wikidata
Bachelorarbeit, November 4, 2019
Reviewers: Prof. Dr. Markus Krötzsch and Prof. Sebastian Rudolph
Supervisors: Prof. Dr. Markus Krötzsch and Prof. Sebastian Rudolph

Technische Universität Dresden
Professur für Wissensbasierte Systeme
Institut für Theoretische Informatik
Fakultät Informatik
01062 Dresden

Kurzfassung

Die freie Wissensdatenbank Wikidata bietet Exporte der Daten als RDF an. Die Größe dieser Exporte stellt jedoch ein Hindernis für die Verarbeitung dar. Auf Basis von Wikidata Toolkit wird in dieser Arbeit eine Anwendung entwickelt, die Dumps für Teile der Daten generiert. Die Anwendung erlaubt es den Nutzern, eigene Kriterien zum Filtern der Exporte anzugeben. Zur Überprüfung der Konsistenz dieser Dumps mit den vollständigen Exporten wird ein Vergleich durchgeführt. Dabei wird das von Wikidata Toolkit erzeugte RDF mit den Wikidata-Exporten verglichen. Mit dieser Methode werden mehrere Fehler in Wikidata Toolkit aufgedeckt. Durch Beheben dieser Fehler leistet die Arbeit einen Beitrag zur Verbesserung des RDF-Exports von Wikidata Toolkit. Mit der entwickelten Anwendung wird die Verwendung von Daten aus Wikidata vereinfacht.

Abstract

Wikidata, the free knowledge base of Wikimedia, provides data exports in RDF. However, usage of these exports is a challenge due to their large size. In this work we present an application based on Wikidata Toolkit that solves this problem by providing dumps of parts of the data. User-defined criteria allow fine-grained filtering of the exports. To verify the consistency of dumps generated this way, we perform a comparision. In this comparision, RDF as generated by Wikidata Toolkit is compared to the full RDF exports provided by Wikidata. Multiple issues in Wikidata Toolkit are uncovered. In addition to developing a useful tool for consumers of Wikidata exports, we contribute several fixes to Wikidata-Toolkit.

Inhaltsverzeichnis

1 Einleitung

Wikipedia ist die bekannteste freie Wissenssammlung im Internet. Bei fast 50 Millionen Artikeln in 278 Sprachen[1] 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. Die Extraktion von Fakten für die maschinelle Verarbeitung aus diesen Textartikeln ist jedoch schwierig [Sch+17; Wan+19]. Das Problem wird durch die unterschiedlichen Sprachversionen noch verstärkt, denn die beschriebenen Fakten können sich je nach Sprache unterscheiden. Als Lösung wurde 2012 das Projekt *Wikidata* gegründet, mit dem Ziel, die Informationen aus Wikipedia in atomare Aussagen zerlegt strukturiert zu verwalten [VK14].

Die Daten aus Wikidata sind für viele Anwendungen aufgrund von zwei Merkmalen besonders interessant. Erstens ist Wikidata ähnlich zu Wikipedia ein Community-Projekt. Die Daten stehen unter einer freien Lizenz und können von jedem ergänzt und korrigiert werden. Deswegen enthält Wikidata Informationen zu einer Vielzahl an Themengebieten. Zweitens besitzt Wikidata ein vielfältiges Datenmodell. Je nach Art der Bestimmung kann beispielsweise entweder der Mount Everest oder Chimborazo als höchster Punkt der Erde gesehen werden. Wikidata kann diese beiden Sichtweisen abbilden, wie Abb. 1.1 zeigt. Um die Herkunft einer Information zu belegen, kann jeder Fakt mit Referenzen versehen werden.

Seit der Gründung ist Wikidata stetig gewachsen. Im September 2019 hat die Größe des vollständigen Exports der Daten als GZip[2]-komprimiertes JSON[3] 60 GB überschritten.[4] Die Größe dieser Datenmenge erschwert die Verarbeitung. Für viele Anwendungen sind aber nicht alle Daten relevant. Ziel dieser Arbeit ist deswegen die Entwicklung eines Systems zum einfachen Abruf einer Teilmenge dieser Daten.

[1] `https://stats.wikimedia.org/DE/TablesArticlesTotal.htm` (alle URLs in dieser Arbeit wurden am 01.10.2019 abgerufen)
[2] `http://www.gzip.org/`
[3] JavaScript Object Notation (`https://www.json.org`)
[4] `https://web.archive.org/web/20190930113616/https://dumps.wikimedia.org/wikidatawiki/entities/`

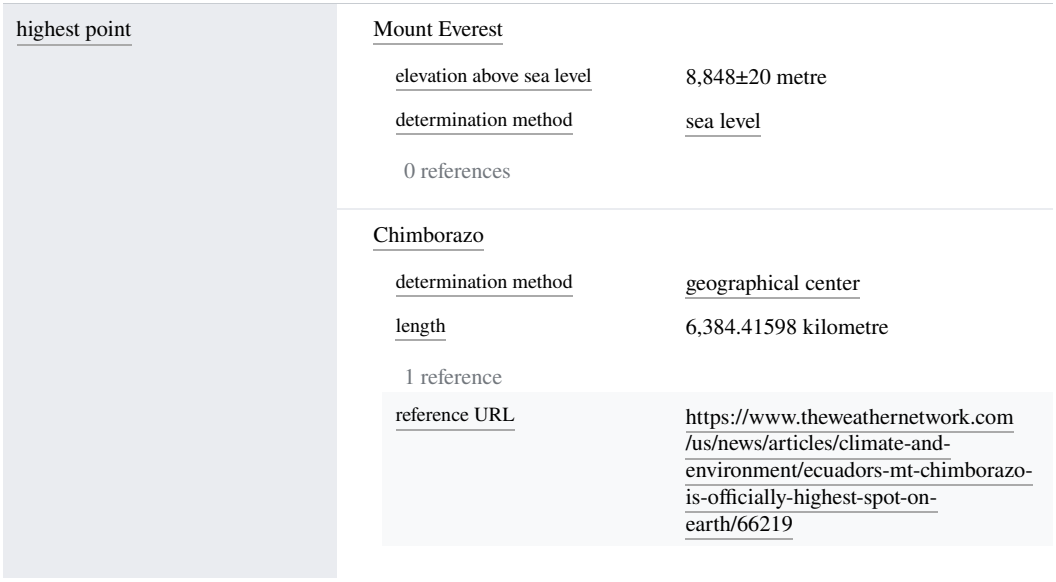

Abbildung 1.1.: Zwei Statements des Items „Erde“ für die Property „höchster Punkt“

Eine bereits existierende Möglichkeit zur Abfrage von Daten ist der Wikidata Query Service [Mal+18]. Mit diesem Dienst lassen sich komplexe Abfragen auf den Daten von Wikidata beantworten. Allerdings ist es mit diesem Dienst sehr schwierig, die kompletten Daten zu einem Thema abzurufen, da das Schema des Query Services die Daten sehr stark zerlegt. Außerdem ist die Laufzeit von Abfragen auf eine Minute limitiert, was bei großen Datenmengen ein Problem ist. Das dieses Limit nicht nur ein theoretisches Problem ist, zeigt eine Arbeit zur Verwendung von Referenzen in Wikidata [Pis+17]. Die Autoren mussten die Abfrage nach allen Referenzen von einem Mitarbeiter von Wikimedia Deutschland ausführen lassen, da sie nicht innherhalb des Limits terminiert.

Das in dieser Arbeit vorgestellte System füllt die Nische zwischen dem vollständigen Export und dem Wikidata Query Service. Durch die Angabe von Filtern wird eine Teilmenge der Daten definiert. Aus den Daten, die diese Filter erfüllen, wird dann ein individualisierter Datenexport (Dump) generiert. Über eine Integration mit dem Archivierungsdienst Zenodo[5] können die generierten Dumps direkt archiviert werden. Zenodo generiert einen Digital Object Identifier (DOI), sodass die Dumps danach in wissenschaftlichen Veröffentlichungen zitiert werden können. Somit wird die Nutzung von Teilenmengen der Daten aus Wikidata deutlich vereinfacht.

[5] `https://zenodo.org/`

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 verwandte Arbeiten verglichen. In Kapitel 4 wird dann das Design des Systems erarbeitet, dessen Implementierung in Kapitel 5 vorgestellt wird. Die Evaluation der Implementierung auf Korrektheit und Vollständigkeit erfolgt in Kapitel 6, mit einem Vergleich zwischen den existierenden Wikidata-Dumps und den generierten Dumps. Im Kapitel 7 wird schließlich eine Zusammenfassung des Ergebnis präsentiert und ein Ausblick auf mögliche Verbesserungen gegeben.

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[6], einer Bibliothek zum Verarbeiten und Erzeugen von Wikidata Dumps.
- Evaluation der Vollständigkeit und Korrektheit des RDF-Exports von Wikidata Toolkit

[6] https://github.com/Wikidata/Wikidata-Toolkit

2 Hintergrund

In diesem Kapitel wird das Datenmodell von Wikidata beschrieben. Danach wird das Format RDF (Resource Description Framework) erläutert und die Abbildung von Wikidata auf dieses Format erklärt. RDF ist Standardformat für den Austausch von strukturierten Informationen.

2.1 Wikidata als strukturiertes Wiki

Wikidata ist die gemeinsame Wissensdatenbank der Wikimedia Projekte. Wie auch Wikipedia selbst basiert Wikidata auf MediaWiki[1], einer Software für das Betreiben von kollaborativen Wikis. An Stelle von Dokumenten in natürlicher Sprache verwaltet Wikidata jedoch strukturierte Dokumente. Die Erweiterungen für MediaWiki stellt dazu das Wikibase[2] Projekt bereit.

Die Dokumentation von Wikibase [Med19] beschreibt das Datenmodell detailliert. 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 *Lexeme* (L). Lexeme wurden erst 2018 hinzugefügt, sind damit noch recht neu. Da Wikidata Toolkit diese noch nicht vollständig unterstützt, werden Lexeme im folgenden nicht genauer betrachtet.

Den Hauptbestandteil der Daten bilden die Items (Q). Items repräsentieren Dinge, die durch Wikidata beschrieben werden. In vielen Fällen existieren auch Wikipedia-Artikel in verschiedenen Sprachen zu dem Ding, welches durch das Item repräsentiert wird. Zur Verlinkung zwischen Item und Seiten des Wikimedia Projekts (Wikipedia-Artikel, Wikiquote-Seiten etc.) zum selben Thema enthält jedes Item eine Liste von *Sitelinks*. Die Properties (P) stellen die Attribute dar, die zur Beschreibung von Items verwendet werden können.

[1] `https://mediawiki.org`
[2] `htts://wikiba.se`

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*. Diese dienen zur Beschreibung und Definition des Items. Die Terme sind mehrsprachig: ein Item kann ein *Label* für viele verschiedene Sprachen haben. Darauf folgen mehrere *Statements*, welche die Fakten zu diesem Item wiedergeben. Den letzten Teil bilden die Sitelinks. Die Sitelinks von Q42 verlinken zum Beispiel unter anderem auf Artikel von 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 Statements ohne Referenzen erlaubt sind, kann die Liste der Referenzen leer sein. Jede Aussage enthält einen *Snak*. Es gibt drei Arten von Snaks:

PropertyValue die am meisten verwendete Art eines Snaks. 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. Eine Liste der möglichen Typen zeigt Tabelle 2.1.

SomeValue diese Art Snak wird verwendet, um auszudrücken, dass ein Wert für die Eigenschaft existiert der aber nicht bekannt ist. Zum Beispiel kann der Snak „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. Diese Art Snak 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* für die Hauptaussage und zusätzlichen *Qualifiern* zur Verfeinerung der Aussage. Eine Referenz ist einfach eine Liste von Snaks.

Ein MainSnak zu Douglas Adams (Q42) ist 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 einen *Rank* (*Deprecated*, *Normal* oder *Best*) um die Priorität innerhalb der Statement Group auszudrücken.

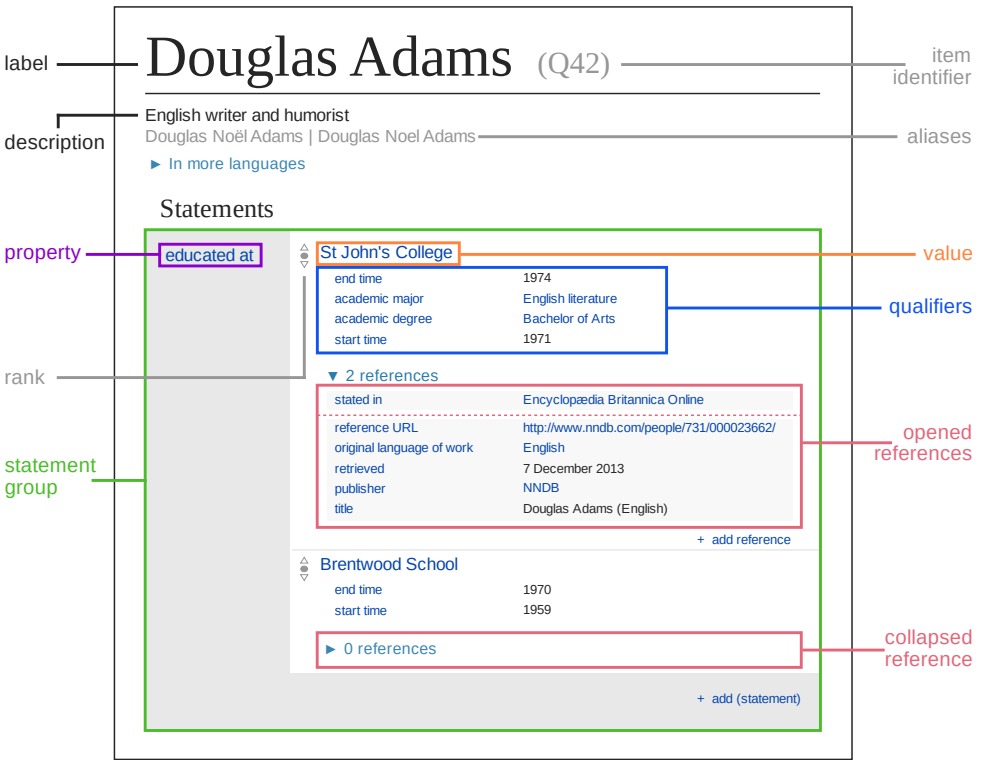

Abbildung 2.1.: Wikidata Item „Q42“.
Quelle: `https://mediawiki.org/wiki/Wikibase/DataModel/Primer`

Dieses dokumentorientierte Datenmodell lässt sich einfach als JSON[3] (JavaScript Object Notation) repräsentieren. Die vollständige Darstellung ist jedoch sehr umfangreich und ist deshalb aus Platzgründen hier nicht abgebildet. Für eine Entität kann die JSON-Repräsentation einfach online abgerufen werden, für `Q42` zum Beispiel unter: `https://www.wikidata.org/wiki/Special:EntityData/Q42.json`.

Wikidata erstellt wöchentlich einen *Gesamtexport* der Daten als JSON-Liste von Dokumenten. Unter `https://dumps.wikimedia.org/wikidatawiki/entities/` lassen sich diese Exporte als GZip[4] (58 GiB[5]) oder BZip2[6] (39 GiB) komprimierte Dateien herunterladen. Der GZip-Export ist zwar etwas größer, dafür ist der Aufwand zur Dekompression geringer, sodass die Geschwindigkeit der Verarbeitung schneller ist.

[3] `https://www.json.org`
[4] `http://www.gzip.org/`
[5] die Zahlen wurden am 18.10.2019 ermittelt
[6] `https://sourceforge.net/projects/bzip2/`

Typ	Beschreibung	Attribute
String	Zeichenkette (ohne Sprachangabe)	nein
Monolingualtext	Zeichenkette (mit Angabe der Sprache)	nein
WikibaseItem	Ein Wikidata Item	nein
WikibaseProperty	Eine Wikidata Property	nein
ExternalId	Ein Bezeichner in einer anderen Datenbank Mit P1630 (formatter URL) wird bei Properties von diesem Typ ein Format zur Erzeugung von URLs aus dem Bezeichner angegeben	nein
Url	Eine URL zur Identifikation einer externen Resource. Beispiele sind Webseiten (`http:`), Email-Adressen (`mailto:`) oder IRC channel (`irc:`)	nein
Math	Eine mathematische Formel; das Format ist ein MediaWiki-spezifisches Subset von LaTeX[a]	nein
MusicalNotation	Eine Sequenz von Noten; Werte mit diesem Typ sind Zeichenketten in der LilyPond[b]-Notation	nein
CommonsMedia	Referenz auf eine Mediendatei in Wikimedia Commons	nein
TabularData	Referenz auf eine Tabelle in Wikimedia Commons	nein
GeoShape	Referenz auf Kartendaten (`.map`) in Wikimedia Commons	nein

Quantity	Eine Zahlenangabe; neben der Zahl selbst (`amount`) kann optional eine untere (`upperBound`) und untere (`lowerBound`) Schranke für das Interval der Unsicherheit angegeben werden; außerdem wird eine Einheit `unit` gespeichert (diese kann für dimensionslose Werte leer sein)	`amount`
		`upperBound`
		`lowerBound`
		`unit`

Time	Ein Zeitpunkt; `time` ist ein Zeitstempel in einem an ISO 8601 angelehnten Format (Beispiel: +2013-01-01T00:00:00Z); `precision` gibt Präzision der Angabe an (100 Millionen Jahre (1), …, Jahrhundert (7), Dekade (8), Jahr (9), Monat (10), Tag (11), …); `before` und `after` geben die Ungenaugkeit der Angabe an (Einheit abhängig von `precision`); `timezone` spezifiziert die Zeitzone, `calendarmodel` das Kalendermodell (momentan werden gregorianisch und julianische Kalender untersüztzt)	`time`
		`before`
		`after`
		`precision`
		`timezone`
		`calendarmodel`

GlobeCoordinate	Geographische Position; `latitute` und `longtitude` geben die Koordinaten an; `globe` spezifiziert den Planet (Standard: `Q2` (Erde)); `precision` speichert die numerische Präzision der Koordinatenangaben; als Koordinatensystem wird das World Geodetic System (WGS84) angenommen	`latitude`
		`longtitude`
		`globe`
		`precision`

WikibaseLexeme	Ein Wikidata Lexem	nein
WikibaseSense	Ein Wikidata Sense (Teil von Lexemen)	nein
WikibaseForm	Wikidata Form (Teil von Lexemen)	nein

[a] `https://en.wikipedia.org/wiki/Help:Displaying_a_formula`
[b] `http://lilypond.org/`

Tabelle 2.1.: Datentypen in Wikidata. Eine Beschreibung dieser Datentypen ist unter `https://www.wikidata.org/w/index.php?title=Help:Data_type&oldid=1027592722` abrufbar

2.2 Resource Description Framework

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 RDF entwickelt [WLC14]. In RDF werden als eindeutige Bezeichner Internationalized Resource Identifiers (IRIs)[7] verwendet. IRIs sind eine Erweiterung von URIs zur Unterstützung von internationalen Zeichensätzen. In der Praxis werden für IRIs oft HTTP 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 dieselben Blank Node Bezeichner verwenden, ohne dass damit dieselbe Ressource beschrieben wird.

Literale in RDF bestehen aus einer Zeichenkette und optional noch einem Datentypen oder einem Language Tag. Mit dem Datentyp kann die Interpretation der Zeichenkette genauer spezifiziert werden. Datentypen werden über IRIs identifiziert. Language Tags geben die Sprache der Zeichenkette als Language Code nach BCP47[8] an. Die Angabe von Language Tags ist nur für Literale mit folgendem Datentyp erlaubt: `http://www.w3.org/1999/02/22-rdf-syntax-ns#langString`.[9]

RDF Dokumente sind eine ungeordnete Kollektion von Tripeln. Für die Serialisierung von RDF gibt es verschiedene Standards; ein einfacher und verbreiteter ist N-Triples [SC14]. Die Syntax von N-Triples ist in Abb. 2.2 dargestellt. Jede Zeile entspricht

[7] https://www.ietf.org/rfc/rfc3987.txt
[8] `https://tools.ietf.org/html/bcp47`
[9] Siehe RDF Spezifikation [WLC14], Sektion 3.3 Literals

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

Abbildung 2.2.: Beispiel für RDF in N-Triples Darstellung

einem Tripel und wird mit einem Punkt abgeschlossen. In N-Triples werden IRIs in <> eingeschlossen und Blank Nodes durch das Präfix `_:` markiert. Literale sind von doppelten Anführungszeichen umgeben, mit einem At-Zeichen (`@`) bzw. doppeltem Zirkumflex (`^^`) 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 Tabelle 2.2 aufgeführten Präfixen vereinfacht. Zum Beispiel wird `<http://schema.org/name>` als `schema:name` abgekürzt.

2.3 Wikidata als RDF

Als Standardformat für Graphdaten ist RDF weit verbreitet. Um existierende Anwendungen mit den Daten von Wikidata verwenden zu können, ist eine Darstellung als RDF sinnvoll. Eine direkte Abbildung von Statements auf Tripel ist jedoch nicht möglich, da RDF das Konzept von Qualifiern und Referenzen nicht kennt. Außerdem erlaubt Wikidata mehrere Statements mit identischem Inhalt, was in RDF nicht vorgesehen ist.

Erxleben, Günther, Krötzsch, Mendez und Vrandečić haben 2014 beschrieben wie trotzdem eine Darstellung als RDF möglich ist und 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 und zusätzlich noch Tripel für Qualifier und Referenzen. Jedes Statement hat somit einen eindeutigen Namen, sodass auch zwei Statements mit demselben Inhalt erfasst werden können.

Das Schema für diese Übersetzung ist in Abb. 2.3 gezeigt. Jeder Pfeil beschreibt dabei Prädikatpräfix, welches für RDF-Tripel verwendet wird. An dieses Präfix wird der Bezeichner der Property angehangen.

Präfix	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 2.2.: Verwendete Präfixe. Alle Präfixe werden auch vom Wikidata Query Service vordefiniert.

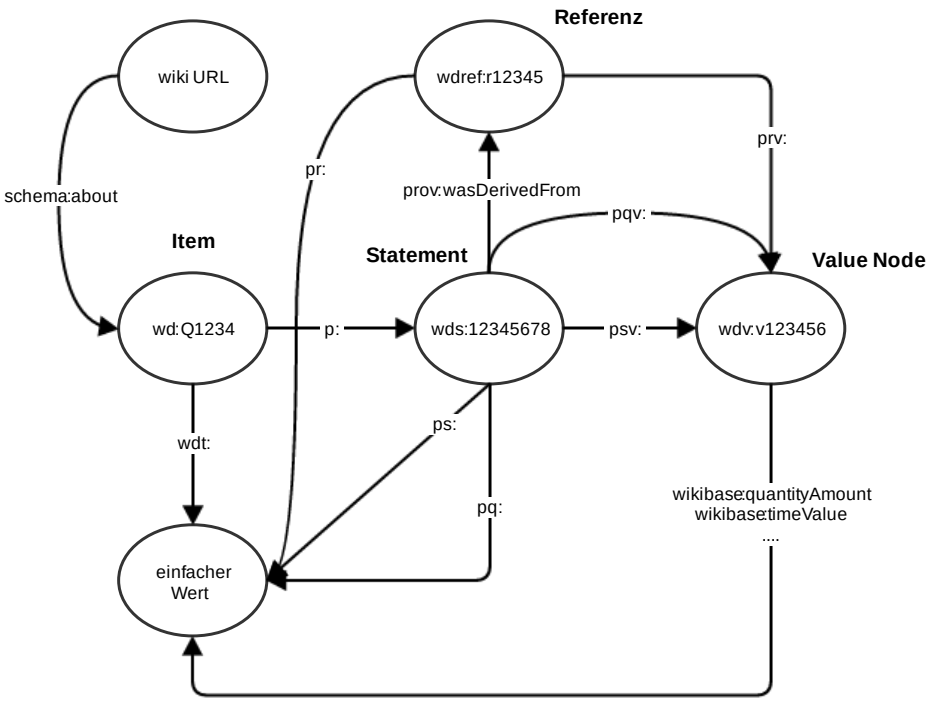

Abbildung 2.3.: Schema zur Übersetzung des Datenmodells nach RDF
Quelle für das Bild: `https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format#/media/File:Rdf_mapping-vector.svg`

Als N-Triples:

```
wd:Q42 p:P26 wds:q42-xxxx .
wds:q42-xxxx rdf:type wikibase:Statement .
wds:q42-xxxx ps:P26 wd:Q14623681 .
wds:q42-xxxx pq:P580 "1991-11-25T00:00:00Z"^^xds:dateTime .
wds:q42-xxxx pqv:P580 wdv:c8ae0d38443d4671d3f893d7000a859e .
wds:q42-xxxx pq:P582 "2001-05-11T00:00:00Z"^^xsd:dateTime .
wds:q42-xxxx pqv:P582 wdv:1c30ade7914d072877b2db404a683d7c .
wds:q42-xxxx prov:wasDerivedFrom wdref:yyyyyyyy .
wd:Q42 wdt:P26 wd:Q14623681 .
```

graphische Darstellung:

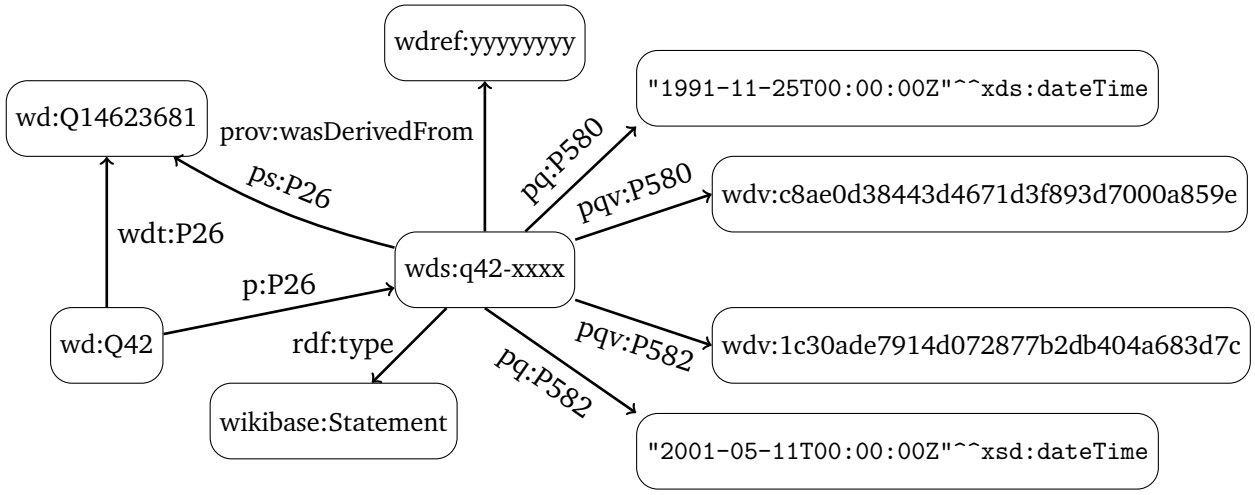

Abbildung 2.4.: Beispiel für die Übersetzung nach RDF

Ein Beispiel für die Übersetzung ist in Abb. 2.4 dargestellt (gekürzt, die Informationen zu den Referenzen und Value Nodes sind nicht abgebildet). Das Beispiel zeigt, wie die Property `P26` (spouse) zuerst mit dem Prädikat `p:P26` auf das Statement verlinkt. Das Statement verlinkt über `ps:P26` zum einfachen Wert der Property. Mit `pq:P580` (start time) bzw. `pqv:P580` wird auf die einfache und komplexe Darstellung (*Value Node*) des Wertes für diesen Qualifier verlinkt. Die Value Nodes werden für Werte verwendet, deren Typ mehrere Attribute hat (siehe Tabelle 2.1). Für Datumsangaben hat die Value Node zum Beispiel Tripel für die Genauigkeit oder das verwendete Kalendermodell.

Eine spezielle Bedeutung hat der `wdt:` Präfix. Tripel mit diesen Prädikaten verlinken Items direkt mit den einfachen Werten. Diese Tripel existieren allerdings nur dann, wenn das zugehörige Statement vom Typ `BestRank` ist. Dieser Typ umfasst alle Statements, die den höchsten Rang für diese Property haben und nicht deprecrated sind. Solche Statements und Tripel werden auch als *truthy* bezeichnet. In vielen Fällen können Abfragen damit deutlich kürzer und zugleich lesbarer formuliert werden, da einfache Verbindungen oft unabhängig von Qualifiern oder Referenzen betrachtet werden.

Eine zweite Vereinfachung sind normalisierte Werte. Diese werden über Prädikate mit dem Präfix `wdtv:`, `psn:`, `pqn:` und `prn:` verlinkt. Bei Werten, die den Typ Quantity haben und eine Einheit besitzen, ist der normalisierte Wert derselbe Wert umgrechnet in Grundeinheiten. Auch bei dem Typ ExternalID können normalisierte Werte auftreten. Für einige Properties, deren Werte den Typ ExternalID haben, speichert Wikidata ein Muster zur Umwandlung des externen Bezeichners in eine URL. Diese URL ist dann der normalisierte Wert.

Auf Basis dieses Schemas hat Wikidata ein offizielles Format für Exporte als RDF entwickelt [10]. Analog zu den Exporten im JSON-Format werden wöchentlich Gesamtexporte aller Daten als RDF erzeugt und zum Download angeboten. Da RDF die Informationen weniger kompakt repräsentiert als JSON, sind die Exporte etwas größer: 76 GiB mit BZip2 und 99 GiB mit GZip. Zusätzlich zu den vollständigen Exporten wird auch ein Export mit nur den truthy Tripeln angeboten, welcher deutlich kleiner ausfällt (32 GiB mit GZip).

Ein bekannter Nutzer des RDF-Formats ist der Wikidata Query Service[11] [Mal+18]. Auf Basis der RDF-Datenbank BlazeGraph[12] wird ein Interface zum Abfragen der

[10] `https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format`
[11] `https://query.wikidata.org`
[12] `https://blazegraph.com/`

Daten mittels SPARQL bereitgestellt. Die Daten werden zu Beginn aus dem RDF-Gesamtexport geladen und dann laufend mit der Wikidata-API inkrementell aktualisiert, sodass immer der aktuelle Stand abrufbar ist.

3 Anforderungen

In diesem Kapitel werden die Anforderungen an das System zum Erstellen von gefilterten RDF-Dumps definiert. Dazu werden zuerst die funktionalen, dann die nichtfunktionalen Anforderungen betrachtet. Um das System von bereits existierenden Systemen abzugrenzen werden danach verwandte Systeme verglichen.

3.1 Funktionale Anforderungen

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

Korrektheit 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 Über eine einfache Oberfläche soll eine Filterung nach mindestens drei Aspekten möglich sein: nach Entität (welche Entitäten sollen exportiert werden), nach Property (Statements für welche Properties sollen exportiert werden) und nach Sprache (in welchen Sprachen sollen Zeichenketten exportiert werden). Für Statements soll auswählbar sein, welche Teile exportiert werden (Qualifier, Referenzen, nur truthy Tripel, vollständiges Statement).

Vollständigkeit Alle Daten, welche den gewählten Filtern entsprechen, sollten im Dump vorhanden sein.

Parallelverarbeitung Mehrere Anfragen sollen parallel verarbeiten werden. Das bedeutet, dass die Zeit zum Erstellen eines Dumps nicht unbegrenzt mit der Anzahl der Nutzer wächst.

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

Suche Es soll eine durchsuchbare Übersicht über alle erstellten Dumps geben. So können Ideen anderer Nutzer als Vorlage dienen. Das spart Zeit, da keine erneute Spezifikation der einzelnen Filter notwendig ist. Außerdem wird damit die Nutzung vorhandener Dumps gefördert, was die Last auf das System verringert und Vergleichbarkeit in Studien verbessert.

Statistiken Um schnell zu entscheiden, ob ein Dump für einen bestimmten Anwendungsfall geeignet ist, sollten Metadaten über den Inhalt (z.B. die Anzahl der Entitäten) des Dumps angezeigt werden.

Fortschritt Während ein Dump generiert wird, soll der Fortschritt des Generierungsprozesses angezeigt 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 nicht notwendig, dass die Dumps immer komplett aktuell sind, aber die Daten zur Generierung der Dumps sollten regelmäßig aktualisiert werden.

Als Grundlage zur Ermittlung der Minimalanforderungen für die Filterkriterien werden existierende Arbeiten betrachtet, welche Exporte von Wissengraphen wie Wikidata gefiltert haben. Ein paar Beispiele dafür sind:

1) nur „truthy“ Statements mit Entitäten als Objekt; Labels/Descriptions 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 festgelegten Properties [Met+19]
5) nur der englische Teil; Labels anderer Sprachen werden verworfen [Has+15]

6) nur Statements für Properties, wobei die Properties Instanzen einer bestimmten Klasse sein müssen, ohne deprecrated, no-value und unknown-value Statements [HMS19] (allerdings wurde der Dump hier nicht als RDF, sondern in der JSON-Form verarbeitet)

7) nur Items von gestorbenen Personen, die an mindestens drei Statements für mindestens 5000 mal vorkommende Properties beteiligt sind [Bor+11]

8) kleinere Datensätze, erzeugt durch eine zufällige Auswahl (*Sampling*) einiger 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 Entitäten (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+17]

Mit den drei beschriebenen Aspekten (Filterung nach Entität, Property und Sprache) und den verschiedenen Optionen für den Export von Statements lassen sich die meisten dieser Anwendungsfälle abdecken. Außerdem fällt auf, dass die Auswahl von Entitäten nur auf lokalen Eigenschaften basiert, die anhand des Entität-Dokuments entschieden werden können. Kompliziertere Kriterien beschränken sich auf die Auswahl der zu exportierenden Statements.

In den genannten initialen Anforderungen ist eine Filterung nach statistischen Merkmalen, wie die Anzahl der Statements einer Entität, und zufällige Auswahl von Objekten (Sampling) noch nicht enthalten. Für einen Prototypen sind diese Funktionen nicht notwendig, denn wie die Liste der Anwendungsfälle zeigt, gibt es auch viele interessante Kriterien die diese Funktion nicht benötigen. Da es jedoch auch mehrere Anwendungsfälle für diese Art der Filterung gibt, sollte das Design um diese Funktionen erweiterbar sein.

3.2 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen setzen sich zusammen aus Standardanforderungen an Softwareprojekte, wie gute Code-Qualität und Dokumentation, und weitere Anforderungen, die sich aus der konkreten Aufgabe und dem Anwendungsfall als Tool im Kontext eines Wikimedia-Projekts ergeben:

Hardwareanforderungen Der Ressourcenverbrauch des Systems sollte in einem akzeptablen Rahmen liegen. Als Anhaltspunkte für die Beurteilung dienen hier vergleichbare Systeme, wie der Wikidata Query Service, welcher ebenfalls Dienste basierend auf den RDF-Daten von Wikidata anbietet und gleichzeitig deutlich populärer ist (44 Anfragen pro Sekunde im Jahr 2018 [Mal+18]). Die Datenbank des Wikidata Query Service war 2018 über 400GB groß. Der Dienst benutzt sechs Server mit folgender Hardware: dual Intel(R) Xeon(R) CPU E5-2620 v4 8 core; 128G RAM; Dual RAID 800G SSD als Speicher. Diese laufen allerdings nicht unter Volllast. Das System sollte entsprechend seiner Relevanz geringere Hardwareanforderungen als der Wikidata Query Service haben. Für die initiale Version des Systems sollten 200GiB Festplattenspeicher und 8 GiB Arbeitsspeicher ausreichen. Diese Anforderungen erlauben das Deployment auf der Wikimedia Toolforge[1]. Damit ist das Deployment für die initiale Version einfach.

Skalierbarkeit Wikidata wächst beständig; Stand September 2019 existieren etwas mehr als 60 Millionen Items[2].

Freie Lizenz Die Wikimedia Foundation legt großen Wert darauf, möglichst viele der verwendeten Tools unter einer freien Lizenz bereitzustellen [Wik13]. Wenn das System in der Wikimedia Cloud betrieben werden soll, dann ist die Veröffentlichung des Quellcodes unter einer freien Lizenz sogar Pflicht [Wik19].

Bearbeitungszeit Das System sollte nicht länger als einen Tag zur Generierung eines Dumps benötigen, besser unter 12 Stunden. Falls die Generierung länger als 10 Minuten dauert, muss die Generierung auf einem Server stattfinden, sodass der Rechner des Nutzers währenddessen ausgeschaltet werden kann.

Erweiterbarkeit Da die funktionalen Anforderungen an das System sehr allgemein sind, muss auf Erweiterbarkeit geachtet werden, sodass neue Anforderungen

[1] Die Toolforge erlaubt nach einer einfachen Registration die Nutzung von verschiedenen Cloud-Services der Wikimedia Foundation: `https://tools.wmflabs.org/`

[2] `https://web.archive.org/web/20190930152246/https://www.wikidata.org/wiki/Wikidata:Statistics`

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. Insgesamt gibt es über 700 Millionen Statements[3] und diese Zahl ist allein im letzten Jahr um 200 Millionen gewachsen. Das System muss mit dieser Menge an Daten umgehen können und dies auch in absehbarer Zukunft noch leisten können.

Code-Qualität Der Quellcode sollte die gängigen Anforderungen an Wartbarkeit und Lesbarkeit erfüllen. Es muss ausreichend Dokumentation geben, dass auch andere Entwickler an dem Projekt weiterarbeiten können.

Dokumentation Neben der Dokumentation des Quellcodes (Punkt Code-Qualität) sollte es auch Dokumentation zur Erklärung der Funktionsweise des Systems geben. Nutzer sollten anhand der Dokumentation die Optionen des Systems verstehen und nachvollziehen können, wie die Dumps erzeugt werden.

Technologien Spezifische Anforderungen an Technologien werden nicht gestellt. Allerdings bietet die Wikimedia-Toolforge als Dienste bereits die Datenbank MySQL und Redis an, sodass die Nutzung dieser Technologien mit weniger Aufwand verbunden ist. Als Programmiersprache sind bekanntere Programmiersprachen vorteilhaft, damit die Wartbarkeit in Zukunft durch andere Personen gesichert ist. Bei Verwendung einer wenig verbeiteten Programmiersprache ist die Warscheinlichkeit höher, dass erst eine Einarbeitungszeit für diese spezielle Programmiersprache erforderlich ist.

3.3 Verwandte Arbeiten

Als verwandte Arbeiten werden an dieser Stelle zuerst die verschiedenen, bereits existierenden Schnittstellen zum Zugriff auf Wikidata diskutiert. Anschließend werden mit DBpedia und HDT zwei Arbeiten betrachtet, welche das Problem der großen Dumps auf andere Art lösen.

Die Wikidata API und der Linked Data Export von Wikidata sind zwei einfache Schnittstellen für den Datenzugriff. Mit dem Linked Data Export können die RDF und JSON Daten einzelner Items über HTTP abgerufen werden. Die API erlaubt das

[3] `https://grafana.wikimedia.org/d/000000175/wikidata-datamodel-statements?refresh=30m&orgId=1`

Abrufen der JSON Daten von bis zu 50 Items in einer Anfrage. Bei beiden Schnittstellen muss vorher bekannt sein, auf welche Items zugegriffen werden soll. Komplexere Kriterien zur Auswahl von Items sind mit dieser Schnittstelle daher nicht möglich. Die Limitation von Anfragen auf wenige Items beschränkt außerdem die maximale Anzahl an Items die exportiert werden können, da sonst zu viele Anfragen notwendig wären.

Eine weitere Art des Datenzugriffs ist mit den JSON- und RDF-Gesamtexporten möglich, die wöchentlich erstellt und von Wikidata zum Download angeboten werden. Für die Verarbeitung der JSON-Exporte und die Generierung von RDF existiert bereits eine Java-Bibliothek, Wikidata-Toolkit.[4] Auch Optionen für die Generierung der RDF-Daten, wie eine Einschränkung auf truthy-Tripel, sind in Wikidata-Toolkit schon implementiert. Wikidata-Toolkit erlaubt so eine einfache Filterung der Dumps. Die Optionen decken jedoch nicht alle Anforderungen ab (die Filterung der zu exportierenden Statements ist zum Beispiel nicht vorgesehen), sodass Wikidata-Toolkit lediglich als Grundlage für die Implementierung des Systems dienen kann. Auch bietet Wikidata-Toolkit weder ein User-Interface noch Möglichkeiten zur Archivierung.

Ein gezielter Zugriff auf die Daten ist mit dem Wikidata Query Service [Mal+18] möglich. Mit der verwendeten Abfragesprache SPARQL [Gro13] lassen sich die in den Anforderungen beschriebenen Filter realisieren. Die Abfrage der gesamten Daten von Items ist mit dem Query Service allerdings schwierig. Da der Query Service auf der RDF-Darstellung von Wikidata basiert, müssen für jedes Item noch explizit die Statements, Values und Referenzen abgefragt werden. Die Konstruktion dieser Abfrage von Hand ist wenig nutzerfreundlich. Des Weiteren ist die Performance dieser Abfrage ein Problem. Bei der Extraktion aller Daten von Items, die Instanz der Klasse Mensch (`Q5`) sind, wird das Timeout der Query Service von einer Minute schon bei weniger als 3000 Items erreicht.[5] Die Archivierung wird bei diesem Ansatz ebenso nicht abgedeckt. Die Daten des Query Services werden live aktualisiert, die Ergebnisse einer Abfrage sind daher zu einem späteren Zeitpunkt verschieden.

Eine zweite Schnittstelle, die auf der RDF-Darstellung basiert, sind Triple Pattern Fragments, welche eine Form von Linked Data Fragments [Ver+16] darstellen. Mit der Triple Pattern Fragments Schnittstelle von Wikidata können einfache Abfragen gestellt werden die nach Tripeln mit einer bestimmten Kombination aus Subjekt, Prädikat und Objekt suchen. Nicht alle Felder müssen angegeben werden, eine Abfragen kann zum Beispiel nach allen Tripeln mit Prädikat `rdfs:label` suchen. Da

[4] `https://github.com/Wikidata/Wikidata-Toolkit`

[5] Die für den Test verwendete Abfrage zeigt Listing A.1 im Appendix. Am 26.10.2019 hat das Limit 3000 ein Timeout verursacht, während die Abfrage mit Limit 2500 innerhalb des Timeouts terminierte.

die Abfragen sehr einfach sind, erfordert die Auswertung dieser Abfragen wenig Ressourcen auf dem Server und es ist keine Laufzeitlimit notwendig. Problematisch ist bei dieser Schnittstelle die Extraktion von Statements, Values und Referenzen. Im Gegensatz zu SPARQL sind Triple Pattern Fragments nicht mächtig genug, um in einer Abfrage alle Tripel eines Items inklusive der Tripel für Statements dieses Items abzufragen. Als Möglichkeiten bleiben nur die Abfrage der Tripel für alle Statements oder die einzelne Abfrage der Tripel jedes Statements. Beide Varianten sind nicht praktikabel, da entweder zu viele Daten abgefragt werden oder zu viele Abfragen notwendig sind.

Für die Auswahl von Entitäten für den Export ist PetScan[6] interessant. Dieses Tool bietet ein User-Interface mit einer Vielzahl an Optionen, um Listen von Wikipedia bzw. Wikidata-Seiten zu erstellen. Das Tool bietet keinen Optionen zum Export der Entitäten als RDF an, es lassen sich damit keine Dumps erstellen. Es liefert jedoch eine Vielzahl an Ideen für mögliche Filterkriterien.

Eine andere Variante, die Größe der Dumps zu reduzieren und damit den Zugang zu erleichern, zeigt DBpedia [Leh+15]. DBpedia partioniert seine Dumps dazu in mehrere, kleinere Dumps. Auch einige aus Wikidata abgeleitete Dumps sind verfügbar[7]. Die Daten in DBpedia stammen aus definierten Extraktionen von Wikimedia Projekten, die jeweils nur bestimmte Relationen extrahieren. Durch das klar definierte Schema ist damit eine Trennung der Datensätze nach der Art der Extraktion möglich. Für Wikidata ist dieser Ansatz deutlich schwieriger, da das Datenschema so flexibel ist und die Relationen (Properties) vorher nicht feststehen. Außerdem gibt es neben den Relationen noch weitere Aspekte, nach denen eine Trennung erfolgen könnte, zum Beispiel ob Statements Referenzen haben oder nicht. Die Definition von Kriterien zur Partionierung von Wikidata ist daher schwierig.

HDT [Fer+13], ein Binärformat für RDF-Daten, verwendet eine optimierte, komprimierte Kodierung zur Speicherung der Dumps. Auf diesem Format lassen sich einfache SPARQL-Abfragen effizient ausführen [MGF12]. HDT-Dumps sind damit kleiner und erlauben trotzdem noch die Filterung nach einfachen Kriterien. 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.

[6] `https://petscan.wmflabs.org/`
[7] `https://databus.dbpedia.org/dbpedia/wikidata/`

4 Design

In diesem Kapitel werden die Anforderungen präzisiert und ein Design für die Umsetzung erarbeitet. Dazu sind mehrere Aspekte relevant: zum einen die technische Umsetzung der Filterung und Generierung von Dumps (Architektur und unterstützte Filterkriterien), zum anderen die Gestaltung des User-Interfaces und die Auswahl externen Dienste (z.B. für die Archivierung der erzeugten Dumps). Diese bedingen sich gegenseitig: Es ist wenig sinnvoll, wenn die Architektur sehr komplexe Filterkriterien unterstützt, dass User-Interface das 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, die gesamte Logik mit einem Client zu implementieren, welcher direkt auf die Wikidata API und den SPARQL-Endpunkt zugreift oder den Dump als Stream verarbeitet. Der Client kann dann von jedem Nutzer lokal auf seinem eigenen System ausgeführt werden. Der Vorteil dieser Variante ist, dass der Server keine eigenen Dumps erzeugen muss, sondern maximal Metadaten zu generierten Dumps speichert. Damit sind wenig Ressourcen für den Betrieb des Servers erforderlich. 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 umgesetzte Anwendung implementiert die Logik zur Generierung der Dumps daher auf dem Server.

Für die Generierung der Dumps auf dem Server sind drei Varianten denkbar:

a) Der Server verwaltet einen eigenen Index, auf dem Filter-Anfragen effizient ausgewertet werden können und das Ergebnis ohne große Kosten als RDF ausgegeben werden kann, ähnlich dem Wikidata Query Service.

b) Der Server nutzt den Wikidata SPARQL Endpunkt und die Wikidata API.

c) Der Server verarbeitet die existierend JSON- oder RDF-Gesamtexporte 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 des gezielten Zugriff auf Teile der Daten hindert außerdem die Kompression. Diese Variante hat damit einen hohen Speicherplatzbedarf.

Variante b) basiert auf der Beobachtung, dass die 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 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 Entitäten):

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

Listing 4.1: SPARQL-Query nach wissenschaftlichen Artikeln

Zu den Items müssten dann auch noch die Statements und Referenzen abgerufen werden, die in diesem Query noch gar nicht betrachtet werden. Durch die Reifikation

von Statements ist zudem die Abfrage aller Daten zu allen Statements bestimmter Items schwierig.

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, denn 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. 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 (zufällige Auswahl eines Subsets der Daten) 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, weil 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.

Das Konzept der gewählten Architektur zeigt Abb. 4.1. Das Frontend stellt das User-Interface bereit, während das Backend die beschriebene sequentielle Verarbeitung implementiert. Die Warteschlange dient als Puffer, sodass das Backend mehrere Aufträge mit einer sequentiellen Verarbeitung des Gesamtexports verarbeiten kann.

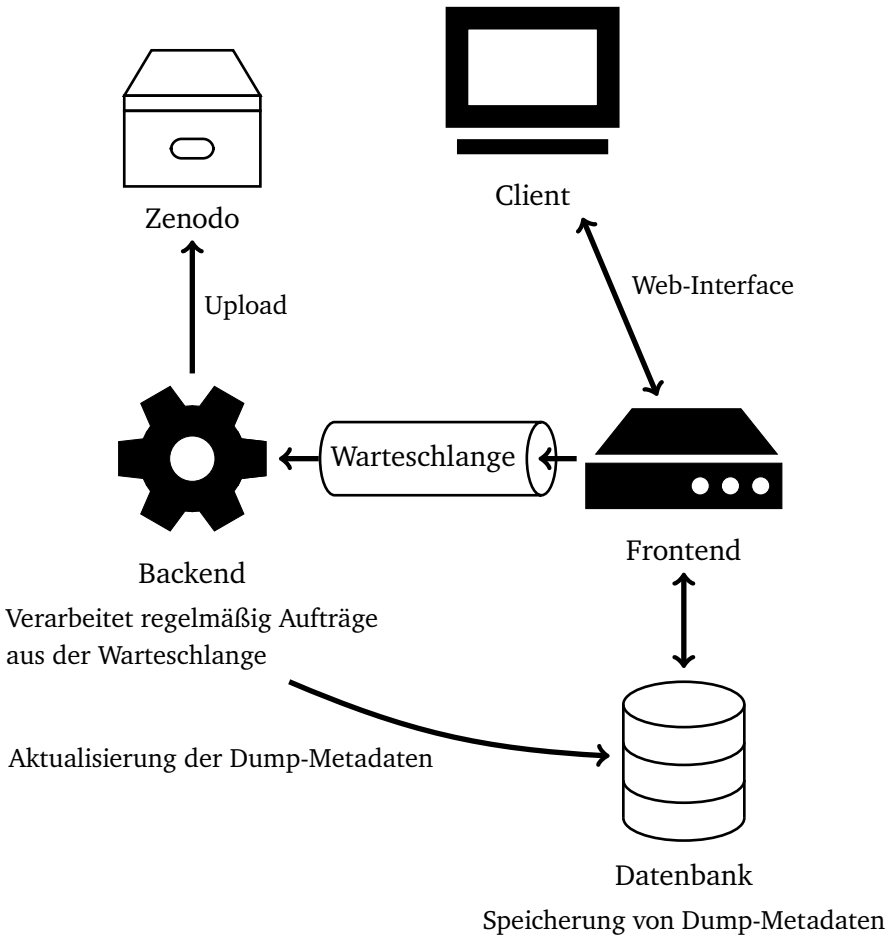

Abbildung 4.1.: Übersicht der gewählten Architektur

4.2 Exportkriterien und 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 an der Struktur der Statements. Beispiele hierfür sind die Analyse von Statements mit mindestens einer Referenz, die Extraktion aller Statements für eine bestimmte Property oder aller Statements mit deprecrated 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 zum Beispiel nur jedes zehnte Statement oder nur jede zehnte Entität zu exportieren.

In Wikidata sind Statements immer ein Teil von Entitäten. Folglich kann der Filterprozess konzeptionell in zwei Schritte zerlegt werden: Zuerst werden die Entitäten ausgewählt, danach für diese Entitäten die zu exportierenden Statements. Zusätzlich gibt es noch weitere globale Optionen, die sich nicht auf einzelne Statements beziehen, wie beispielsweise welche Sprachen exportiert werden sollen oder ob Sitelinks von Entitäten exportiert werden sollen. Diese Struktur des Interfaces ist in Abb. 4.2 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,

Abbildung 4.2.: Struktur des Interfaces zum Erstellen von Dumps

keinen Filter anzugeben. In diesem Fall werden alle Entitäten exportiert (entsprechend der 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 Prädikate 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. Dafür sind Schalter für den Export von einfachen Werten, komplexen Werten, normalisiert oder nicht normalisiert möglich. Insgesamt ist dieser Teil einfach erweiterbar, da die Optionen alle binär und somit einfach umsetzbar sind.

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 Sprachen 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 Toolforge[1] an. Die

[1] `https://tools.wmflabs.org/`

Toolforge hat eine Grid-Engine, die zur Ausführung von längeren Jobs wie dem Verarbeiten der Dumps geeignet ist, eine MariaDB-Datenbank und einen Dienst zum Betreiben von Web-Services. Die Alternative zur Toolforge wäre ein Wikimedia Cloud VPS Projekt[2]. 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 Zenodo[3] vorgesehen. Zenodo ist ein Datenarchivierungsdienst speziell für wissenschaftliche Zwecke und besitzt auch die Möglichkeit, einen Digital Object Identifier (DOI) zu registrieren, womit 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 die Nutzer selbst einen Account bei Zenodo benötigen.

[2] `https://wikitech.wikimedia.org/wiki/Portal:Cloud_VPS`
[3] `https://zenodo.org/`

5 Implementierung

Die Anwendung besteht aus zwei Teilen: Backend und Frontend. Der Hauptteil der Anwendung läuft auf einem Server. Das Frontend hat eine JavaScript-Komponente für das User-Interface, welche auf dem Client ausgeführt wird. 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 Design der Anwendung verwendet eine Warteschlange zur Kommunikation zwischen Frontend und Backend. Für die Implementierung wird diese Warteschlange über eine gemeinsame Datenbank realisiert, die bereits vom Frontend zur Verwaltung der Metadaten der Dumps eingesetzt wird. In diesen Kapitel wird erst das Datenmodell vorgestellt, welches zur Kommunikation verwendet wird. Danach wird die Implementierung von jeweils Frontend und Backend genauer beschrieben.

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 (`completed_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. Die ungefähre Anzahl der Entitäten in dem vollen Wikidata-Dump ist aus vorherigen Durchläufen bekannt oder kann über den Wikidata Query Service mit einer Abfrage ermittelt werden. Zusammen mit der Anzahl an bereits verarbeiteten Entitäten lässt sich daraus der Fortschritt errechnen. Die `started_at` und `finished_at` Attribute speichern Start- und Endzeitpunkt des `Runs`. Für noch nicht abgeschlossene `Runs` hat `finished_at` den Wert `NULL`.

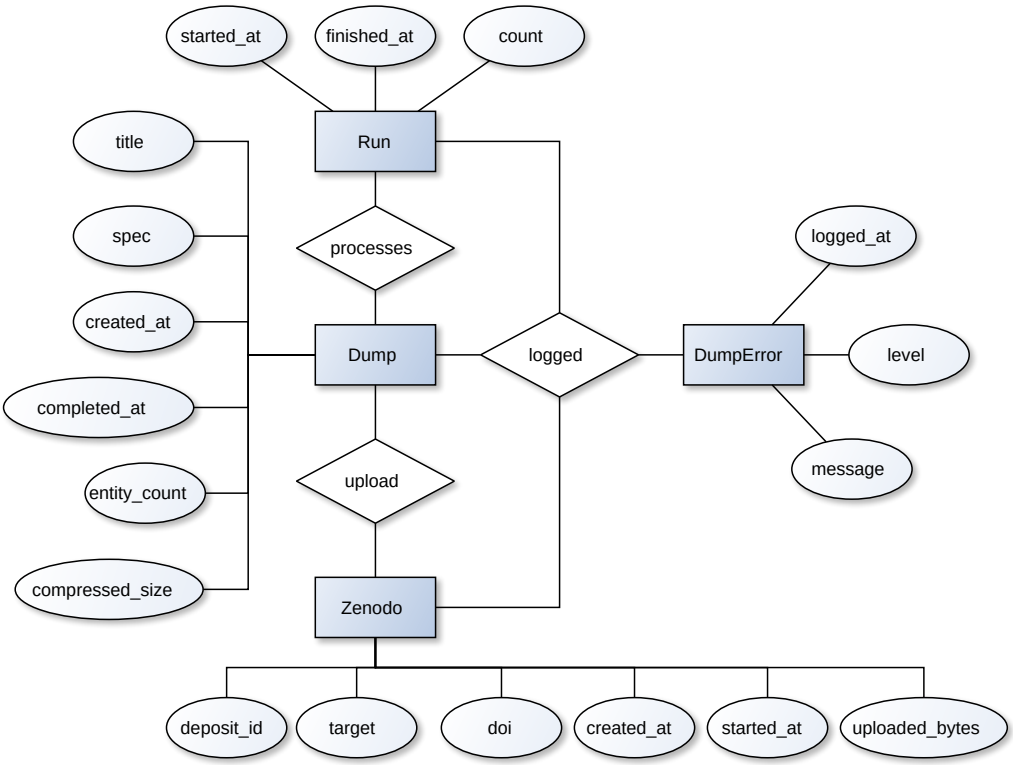

Abbildung 5.1.: Datenbankschema

In der Tabelle `Zenodo` werden Uploads von Dumps verwaltet. Für Uploads muss bei Zenodo ein Deposit angelegt werden. Den Bezeichner dieses Deposits speichert `deposit_id`. Mit `target` (`RELEASE` oder `SANDBOX`) wird die Zenodo-Instanz angegeben. `RELEASE` bezeichnet die Hauptinstanz, während `SANDBOX` eine Instanz zum Testen ist. Der von Zenodo generierte Digital Object Identifier wird in dem Attribut `doi` gespeichert. Zur Verwaltung und Anzeige des Fortschritts werden Zeitdaten (`created_at` und `started_at`) und die Anzahl der bis jetzt hochgeladenen Bytes gespeichert (`uploaded_bytes`).

Für Fehlermeldungen bei der Verarbeitung gibt es die Tabelle `DumpError`. Fehlermeldungen können mit einem bestimmten `Run`, `Dump` und `Zenodo` (Upload) verknüpft werden. Jeder Fehler hat ein `level` (`CRITICAL`, `WARNING`, ...), einen Zeitstempel (`logged_at`) und eine Meldung (`message`).

5.2 Backend

Für die Implementierung des Backends wird das Wikidata Toolkit[1] 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 Scala[2] oder Kotlin[3]) 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 periodisch die in Listing 5.1 dargestellten SQL-Befehle 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 gleichzeitg ausgeführt werden ohne dabei Kollisionen zu erzeugen. Es ist so unmöglich, dass ein Auftrag mehr als einem `Run` zugwiesen wird, was die Robustheit des Systems erhöht. Falls diese Befehle keine Ergebnisse liefern (wenn keine

[1] `https://github.com/Wikidata/WikidataToolkit`
[2] https://www.scala-lang.org/
[3] `https://kotlinlang.org/`

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.

```
-- neuen Run erstellen
INSERT INTO run () VALUES ()
-- generated id: 1

-- Aufträge dem Run zuweisen
UPDATE dump SET run_id = 1 WHERE run_id IS NULL

-- Zugewiesene Aufträge abrufen
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, welche das von Wikidata Toolkit erwartete Interface `EntityDocumentProcessor` implementiert (Listing 5.2).

```
  public interface EntityDocumentProcessor {
    void processItemDocument(ItemDocument itemDocument);
    void processPropertyDocument(PropertyDocument propertyDocument);
    void processLexemeDocument(LexemeDocument lexemeDocument);
  }
```

Listing 5.2: EntityDocumentProcessor Interface

Listing 5.3 zeigt den Ablauf des Exports in Pseudocode. Für jede Entität (Items, Properties und Lexeme) wird dazu zunächst überprüft, ob sie 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 Lexeme noch nicht untersützt, da Wikidata Toolkit den RDF-Export dafür noch nicht implementiert hat. Wenn ein Lexem exportiert werden soll wird deshalb ein Fehler erzeugt. Dieser Fall sollte nicht auftreten, da die Filter-Spezifikation vorgibt, das Lexeme nicht exportiert werden. Zusätzlich zu den Statements wird noch RDF für Labels, Descriptions, Aliases, Sitelinks und Metadaten zu der Entität erzeugt, falls von der Filter-Spezifikation verlangt.

```
for each entity:
  if spec includes entity:
    if entity is lexeme: raise error

    if spec.labels: export entity labels
```

```
    if spec.aliases: export entity aliases
    if spec.descriptions: export entity descriptions

    for each statement:
      options = get options for statement from spec
      export statement with options

    if spec.sitelinks: export entity sitelinks
    if spec.meta: export entity metadata
```

Listing 5.3: Pseudocode für Entity-Export

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. Ein `EntityDocumentProcessor` zählt die Anzahl der verarbeiteten Entities mit und speichert diese regelmäßig in der `run`-Tabelle der Datenbank.

Das Interface zur Angabe von Optionen für einen Dump zeigt Listing 5.4. Wie bereits in Kapitel 4 beschrieben lässt sich die Spezifkation in drei Aspekte zerlegen. Zuerst wird geprüft, ob ein Dokument exportiert werden soll (Methode `includeDocument`). Danach werden für jedes Statement in diesem Dokument die Optionen bestimmt (`findStatementOptions`). Der dritte Aspekt sind globale Optionen, wie die zu exportierenden Sprachen (`includeLanguage`) und einige binäre Optionen.

```
public interface DumpSpec {
  public boolean includeDocument(StatementDocument doc);

  public StatementOptions findStatementOptions(String property);

  public boolean includeLanguage(String code);
  public boolean isSitelinks();
  public boolean isLabels();
  public boolean isDescriptions();
  public boolean isAliases();
  public boolean isMeta();
}

public interface StatementOptions {
  public boolean isSimple();
  public boolean isFull();
  public boolean isReferences();
  public boolean isQualifiers();
}
```

Listing 5.4: Interface für die Filterkriterien

Die Anwendung realisiert dieses Interface auf Basis einer Spezifikation im JSON-Format, die vom Frontend generiert wird.

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 wird eine minimale serverseitige Anwendung verwendet, welche in Python mit Flask geschrieben ist.

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

- `GET /` liefert die Startseite der Anwendung mit Interface zum Erstellen von Dumps aus
- `POST /create` erstellt einen neuen Dump-Auftrag. Im Request-Body werden die Filter-Spezifikation sowie ein paar Metadaten (Titel, etc.) als JSON 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.
- `POST /zenodo` fordert Upload eines Dumps zu Zenodo an (Details werden als JSON-Body übergeben)

Diese Endpunkte implementieren wenig Anwendungslogik, sondern dienen nur als Schnittstelle zwischen dem Frontend und dem Backend. Dazu werden Information in die Datenbank eingetragen oder abfragt.

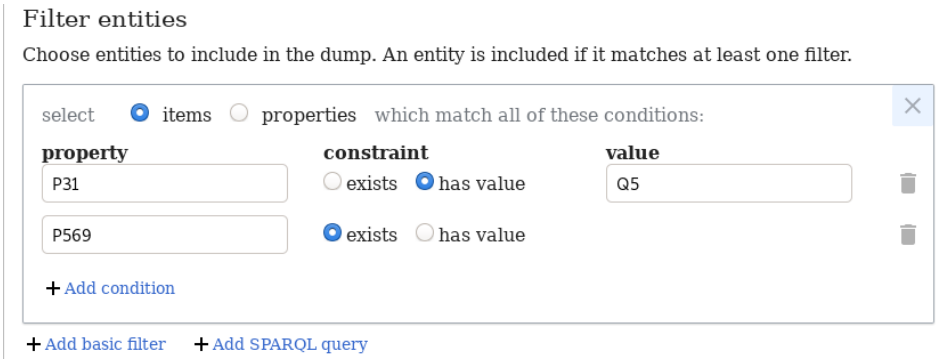

Abbildung 5.2.: Sektion zum Filtern; Im Beispiel werden nur Instanzen von (P31) Mensch (Q5) exportiert, die ein Statement für P569 (Geburtsdatum) besitzen.

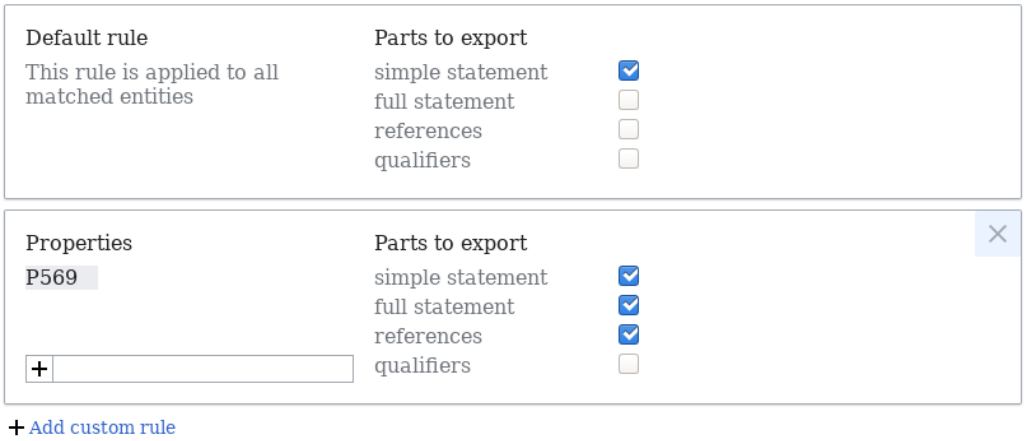

Abbildung 5.3.: Sektion mit Statementeinstellungen; Im Beispiel: Export aller simple Statements, dazu die Statements für P569 (Geburtsdatum) mit Referenzen

5.4 Ergebnis

Die Implementierung umfasst 1600 Zeilen Java (Backend), 200 Zeilen Python (Frontend) und 700 Zeilen TypeScript (Frontend).[4]. Screenshots der wesentlichen Teile der Anwendung zeigen Abb. 5.4, Abb. 5.3 und Abb. 5.2. Zusätzlich zu den gezeigten Ausschnitten gibt es noch eine einfache Liste der bereits generierten Dumps (mit Titel). Für jeden Dump existiert eine Statusseite, die Auskunft über den Fortschritt der Generierung und den Inhalt des Dumps liefert. Die Anwendung ist online unter der URL `https://tools.wmflabs.org/wdumps/` erreichbar. Der Quellcode ist unter MIT-Lizenz auf GitHub veröffentlicht: `https://github.com/bennofs/wdumper`. Für die Anwendungen wurden auch Verbesserungen an Wikidata Toolkit vorgenommen. Diese sind mittlerweile in das offiziele Wikidata Toolkit Projekt eingeflossen.

[4]Ermittelt mit dem Tool cloc, ohne Kommentare (`https://github.com/AlDanial/cloc`)

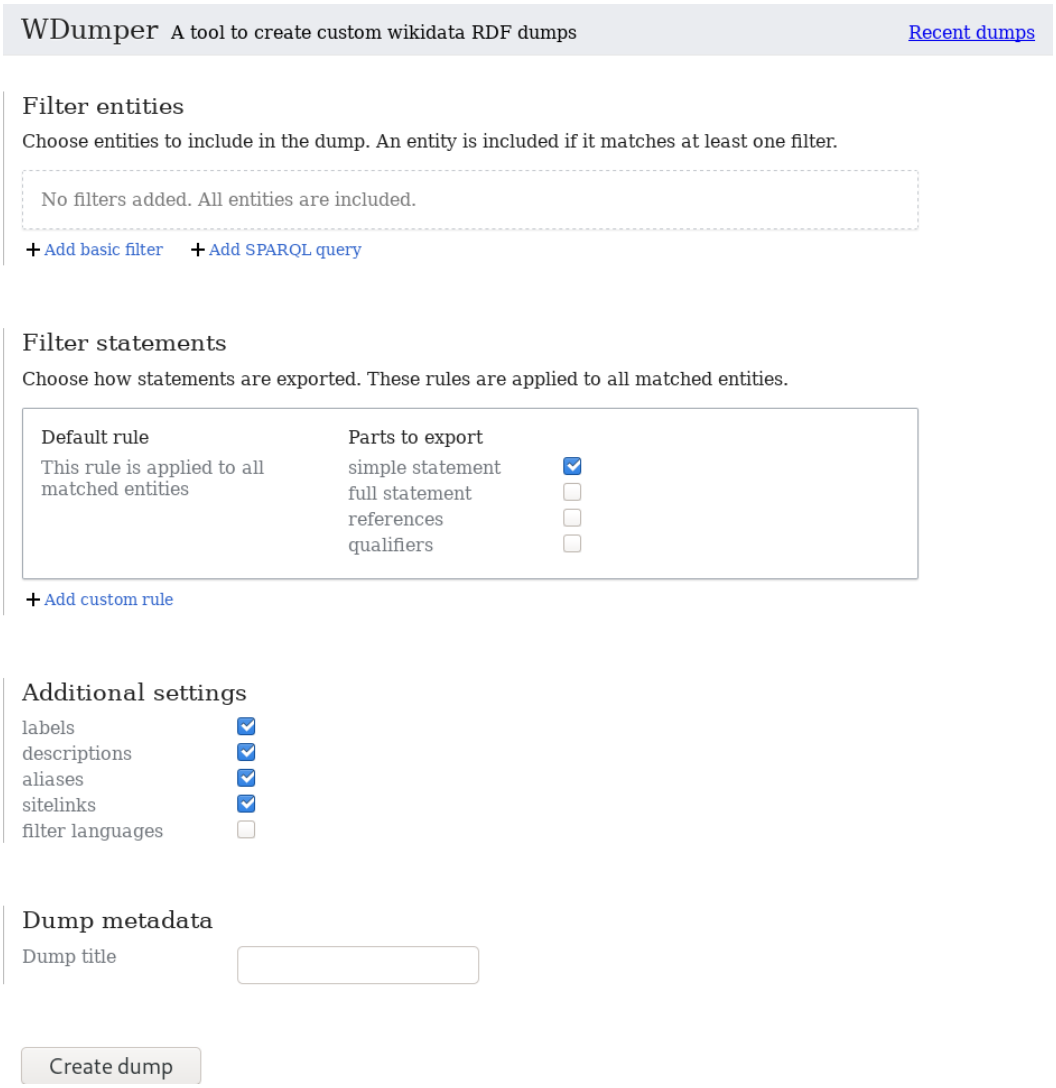

Abbildung 5.4.: Startseite mit Interface zur Angabe von Filtern. In der gezeigten Standardeinstellung werden die simple Statements, Terme und Sitelinks zu jeder Entität in allen Sprachen exportiert.

6 Evaluation

In diesem Kapitel wird die Umsetzung der Anwendung bewertet. Viele der Anforderungen wurden bereits im Design beachtet. Zur Evaluation wird deshalb nur ein Aspekt genauer betrachtet: Die Korrektheit und Vollständigkeit der Dumps.

Die umgesetzte Anwendung verwendet Wikidata Toolkit zum Erzeugen der Dumps. Die vollständigen RDF-Exporte von Wikidata hingegen werden von Wikibase, einer PHP-Erweiterung für MediaWiki, generiert. Damit kann es zwischen beiden Implementierung zu Differenzen kommen. Um diese Differenzen zu finden und zu beheben wird in diesem Kapitel ein offizielle RDF-Export von Wikidata mit den von Wikidata Toolkit generierten RDF Daten verglichen.

6.1 Vorgehen

Für den Vergleich werden die JSON- und RDF-Exporte von Wikidata vom 26. August 2019 verwendet. Die Daten und der Quellcode für den Vergleich sind über Zenodo verfügbar [Fün19]. Da sich die Daten während der Generierung des Exports verändern, beschreiben die JSON- und RDF-Exporte nicht exakt dieselben Daten. Deswegen werden für den Vergleich zusätzlich noch die Wikidata incremental Dumps verwendet. Diese enthalten die JSON-Darstellung 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 im RDF-Export enthaltene 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 dieselbe 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 im JSON-Export und im RDF-

Export in derselben 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 exakt 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. Die Abbildung lässt sich leicht bestimmen, 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 zugeordnet werden. Falls die Abbildung trotzdem noch uneindeutig ist, werden auf beiden Seiten die Menge der Objekte der zu den Value Nodes gehörenden Tripel bestimmt. Danach wird die Abbildung so gebildet, dass möglichst wenig Unterschiede in den Objektmengen der zugeordneten Value Nodes auftreten. 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 Metadaten zu der Entität, wie `schema:dateModified`.

2. Entitäten, welche nur im RDF-Export oder nur im JSON-Export enthalten sind. Dieser Fall kann aus zwei Gründen auftreten: Löschungen in der Zeit zwischen der Erstellung beider Exporte, oder Weiterleitungen. Die RDF-Exporte von Wikidata enthalten für jede Weiterleitung ein Tripel mit Prädikat `owl:sameAs`, welche Wikidata Toolkit nicht generiert.

3. Tripel, die normalisierte Werte für Properties angeben und die entsprechenden Value Nodes. Wikidata Toolkit unterstützt normalisierte Werte 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. Tripel mit den Prädikaten `owl:onProperty` oder `owl:complementOf` sowie Tripel mit Objekten aus dieser Liste: `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 die 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, Datumdeduplizierung und Referenzdeduplizierung. Während der Arbeit wurden diese Korrekturen in Wikidata Toolkit implementiert und sind inzwischen auch Teil des offiziellen Wikidata Toolkit Projekts. 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). Entsprechend der GeoSPARQL Spezifikation [12] ist für den Datentyp `geo:wktLiteral` die Reihenfolge der Koordinaten abhängig vom angegebenen Koordinatensystem. Für den Standardwert CRS84[1] ist die Reihenfolge *Längengrad-Breitengrad*, was entgegen der verbreiteten Darstellung als `50°38'28"N, 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 dieselben 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

[1] `http://www.opengis.net/def/crs/OGC/1.3/CRS84`

Stichwort	**Abweichung in Wikidata Toolkit**
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)	Bei 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 Exponentenschreibweise
Kalender	Umrechnung von Datumsangaben in gregorianischen Kalender fehlt
invalide Datumsangaben	Mehrere Abweichungen bei invaliden Daten (30 Februar etc)
Commons Link	Links zu Wikimedia Commons sind unterschiedlich (falsches Prefix und Behandlung von Sonderzeichen)
mathematische Formeln	Konvertierung nach MathML fehlt
falsche Literaltypen	Tippfehler bei `wikibase:GlobeCoordinatesValue`, fehlender Datentyp `wikibase:MusicalNotation`
Rundung	Gleitkommazahlen unterscheiden sich aufgrund von Rundungsfehlern
Datumdeduplizierung	Datumsangaben werden 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)

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.

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:

```
# tripel aus dem Wikidata RDF Export
wd:Q45 wdt:P571 "1143-10-12T00:00:00Z"^^xsd:dateTime .

# tripel von Wikidata Toolkit generiert
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 BCP47[3] 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

[2] `https://www.w3.org/TR/xmlschema11-2/#dateTime`
[3] `https://tools.ietf.org/html/bcp47`

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. Da die Wikimedia Language Codes jedoch nicht mit BCP47 kompatibel sind, wurde für diese Abweichung das Verhalten in Wikidata Toolkit nicht geändert.

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

7 Fazit

Zum Abschluss der Arbeit wird in diesem Kapitel eine Zusammenfassung der Ergebnisse präsentiert. Die wesentlichen Anforderungen konnten erfüllt werden. Aufgrund der großen Breite des Themas gibt es jedoch immer noch viele mögliche Erweiterungen für die entwickelte Anwendung. Einige dieser zukünftigen Erweiterungen und Verbesserungen werden im folgenden Ausblick diskutiert.

7.1 Zusammenfassung

Die Arbeit hat demonstriert, dass das entwickelte Design erfolgreich umgesetzt werden kann. Damit wurde das anfangs gestellte Problem, kleinere Dumps für Teilmengen der Daten aus Wikidata zu erzeugen, gelöst. Der Vergleich der erzeugten Dumps mit den direkten RDF-Exporten von Wikidata hat einige Differenzen aufgedeckt. Die Fehler in Wikidata Toolkit konnten behoben werden, sodass die Korrektheit sichergestellt ist. Nicht behoben werden konnten einige kleinere Differenzen, wie zum Beispiel Abweichungen bei Fließkommazahlen. Diese entstehen durch fehlende Informationen in den JSON-Exporten, daher gibt es keine Möglichkeit, diese in Wikidata Toolkit zu korrigieren. Die generierten Dumps werden dadurch jedoch nur marginal beeinflusst, sodass es in der Praxis kein Problem darstellt.

Von den Verbesserungen in Wikidata Toolkit profitiert nicht nur die entwickelte Anwendung. Die Änderungen wurden dem offiziellen Wikidata Toolkit Projekt beigetragen. Andere Anwendungen, die ebenfalls auf Wikidata Toolkit basieren, sind damit eingeschlossen.

Besonders bei den Filterkriterien ist es schwierig, im Vorfeld alle Möglichkeiten zu implementieren. Mit der entwickelten Anwendung, die online verfügbar ist, kann nun hilfreiches Feedback gesammelt werden. Auf Basis dieses Feedbacks können dann weitere Features implementiert werden.

7.2 Ausblick

Die Möglichkeiten zur weiteren Verbesserung der Anwendung lassen sich in drei Kategorien einteilen: User-Interface, Performance und weitere Filterkriterien. Das User-Interface ist besonders im Bereich des Entdeckens und der Anzeige schon generierter Dumps noch ausbaufähig. In der aktuellen Version gibt es nur eine Liste der Dumps. Für den Anfang ist das aufgrund der geringen Anzahl an generierten Dumps jedoch kein Problem. Zur Beschreibung der Dumps wäre mindestens noch ein weiteres Eingabefeld, zur Eingabe eines längeren Texts, sinnvoll. Der Eintrag bei Zenodo wird mit dieser zusätzlichen Information ebenfalls aussagekräftiger. Auch eine automatische Vorschau ist denkbar.

Eine spannende Variante zur Verbesserung der Performance ist die Parallelisierung. Die Generierung der Dumps sollte sich dazu gut eignen, da jedes Dokument einzeln und unabhängig von den anderen Dokumenten verarbeitet wird. Das Problem ist hier die Komprimierung. Auf der einen Seite sind die Eingabedaten komprimiert, sodass nicht einfach an einer beliebigen Stellen mit der Verarbeitung angefangen werden kann. Auf der anderen Seite müssen auch die Ausgabedaten wieder komprimiert werden, sodass diese nicht vollständig unabhängig voneinander generiert werden können. Für die Eingabe lässt sich das Problem lösen, indem der BZip2[1]-Kompressionsalgorithmus verwendet wird, da dieser Block-basiert ist. Somit ist eine unabhängige, parallele Verarbeitung mehrerer Blöcke möglich. Für die Ausgabe muss eine Möglichkeit gefunden werden, die verschiedenen Teile unter Beibehaltung der Kompression zusammenzuführen. Die Parallelisierung sollte die Performance nahezu linear verbessern. Mit zwei statt einem Prozess würde die Generierung dann nur noch halb so lange dauern.

Weitere Erweiterungsmöglichkeiten bieten sich bei den Filterkriterien. Aufwändig ist dabei vor allem die Umsetzung im User-Interface. Für neue Filter muss darauf geachtet werden, wie diese den Nutzern verständlich präsentiert werden. Ideen für Filter gibt es viele, wie zum Beispiel zufällige Auswahl (Sampling) des Ergebnis zur Reduzierung der Datenmenge.

[1] `https://sourceware.org/bzip2/`

Literatur

[BM19] Rajarshi Bhowmik und Gerard de Melo. “Be Concise and Precise: Synthesizing Open-Domain Entity Descriptions from Facts”. In: *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. Hrsg. von Ling Liu, Ryen W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo Baeza-Yates und Leila Zia. ACM, 2019, S. 116–126 (siehe S. 17).

[BM18] Rajarshi Bhowmik und Gerard de Melo. “Generating Fine-Grained Open Vocabulary Entity Type Descriptions”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*. Hrsg. von Iryna Gurevych und Yusuke Miyao. Association for Computational Linguistics, 2018, S. 877–888 (siehe S. 17).

[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).

[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, Chris Welty, Craig A. Knoblock, Denny Vrandečić, Paul T. Groth, Natasha F. Noy, Krzysztof Janowicz und Carole A. Goble. Bd. 8796. LNCS. Springer, 2014, S. 50–65 (siehe S. 9).

[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. 21).

[FI19] Nuno Freire und Antoine Isaac. *Technical usability of Wikidata’s linked data – Evaluation of machine interoperability and data interpretability*. 2019. URL: https://pdfs.semanticscholar.org/f6d1/6eaf975af03a172c73843ff506592c952a04.pdf (besucht am 28. Okt. 2019) (siehe S. 16).

[Fün19] Benno Fünfstück. *Wikidata-Toolkit RDF diff comparision*. Okt. 2019 (siehe S. 38).

[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).

[Gri18] Hillary K. Grigonis. *TODO*. 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).

[Gro13] W3C SPARQL Working Group. *SPARQL 1.1 Overview*. W3C Recommendation. W3C, März 2013 (siehe S. 20).

[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: *Formal Concept Analysis - 15th International Conference, ICFCA 2019, Frankfurt, Germany, June 25-28, 2019, Proceedings*. Hrsg. von Diana Cristea, Florence Le Ber und Baris Sertkaya. Bd. 11511. LNCS. Springer, 2019, S. 315–323 (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. 16).

[Leh+15] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer und Christian Bizer. "DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia". In: *Semantic Web* 6.2 (2015), S. 167–195 (siehe S. 21).

[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, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee und Elena Simperl. Bd. 11137. LNCS. Springer, 2018, S. 376–394 (siehe S. 2, 13, 18, 20).

[MGF12] 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 - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*. Hrsg. von Elena Simperl, Philipp Cimiano, Axel Polleres, Óscar Corcho und Valentina Presutti. Bd. 7295. LNCS. Springer, 2012, S. 437–452 (siehe S. 21).

[Med19] MediaWiki contributors. *Wikibase/DataModel – MediaWiki*. 2019. URL: https://www.mediawiki.org/w/index.php?title=Wikibase/DataModel&oldid=3250513 (besucht am 12. Okt. 2019) (siehe S. 4).

[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. 16).

[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. 16).

[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, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Fridman Noy und Eva Blomqvist. Bd. 7031. LNCS. 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. 16).

[12] *OGC GeoSPARQL - A Geographic Query Language for RDF Data*. OGC 11-052r4. Open Geospatial Consortium. 2012 (siehe S. 40).

[Pis+17] Alessandro Piscopo, Pavlos Vougiouklis, Lucie-Aimée Kaffee, Christopher Phethean, Jonathon S. Hare und Elena Simperl. "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. 2, 17).

[Sch+17] Rudolf Schneider, Tom Oberhauser, Tobias Klatt, Felix A. Gers und Alexander Löser. "Analysing Errors of Open Information Extraction Systems". In: *Proceedings of the First Workshop on Building Linguistically Generalizable NLP Systems*. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, S. 11–18 (siehe S. 1).

[SC14] Andy Seaborne und Gavin Carothers. *RDF 1.1 N-Triples*. W3C Recommendation. http://www.w3.org/TR/2014/REC-n-triples-20140225/. W3C, Feb. 2014 (siehe S. 8).

[Ver+16] Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck und Pieter Colpaert. “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. 20).

[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).

[Wan+19] Haoyu Wang, Ming Tan, Mo Yu, Shiyu Chang, Dakuo Wang, Kun Xu, Xiaoxiao Guo und Saloni Potdar. “Extracting Multiple-Relations in One-Pass with Pre-Trained Transformers”. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Hrsg. von Anna Korhonen, David R. Traum und Lluís Màrquez. Association for Computational Linguistics, 2019, S. 1371–1377 (siehe S. 1).

[Wik13] 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. 18).

[Wik19] 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. 18).

[WLC14] David Wood, Markus Lanthaler und Richard Cyganiak. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. W3C, Feb. 2014 (siehe S. 8).

Appendix

A

SPARQL-Abfrage zum Test des Timeouts (terminiert für Limit 2500, verursacht ein Timeout bei Limit 3000):

```
CONSTRUCT {
  ?node ?p ?o .
}

# get items
WITH {
  SELECT ?item WHERE {
   # instances of class human
   ?item wdt:P31 wd:Q5 .
  } LIMIT 3000
} AS %items

# get statements for items
WITH {
  SELECT ?stmt WHERE {
    INCLUDE %items .
    ?item ?p ?stmt .
    [] wikibase:claim ?p .
  }
} AS %stmts

# get references for statements
WITH {
  SELECT REDUCED ?ref WHERE {
    INCLUDE %stmts .
    ?stmt prov:wasDerivedFrom ?ref .
  }
} AS %refs

# get values for statements/references
WITH {
  SELECT REDUCED ?val WHERE {
    {
      INCLUDE %stmts .
      ?stmt ?p ?val .
      [] wikibase:statementValue|wikibase:statementValueNormalized|
```

```
          wikibase:referenceValue|wikibase:referenceValueNormalized|
          wikibase:qualifierValue|wikibase:qualifierValueNormalized
        ?p .
      FILTER (STRSTARTS(STR(?val), "http://www.wikidata.org/value/"))
    }
    UNION
    {
      INCLUDE %refs .
      ?ref ?p ?val .
      [] wikibase:statementValue|wikibase:statementValueNormalized|
         wikibase:referenceValue|wikibase:referenceValueNormalized|
         wikibase:qualifierValue|wikibase:qualifierValueNormalized
        ?p .
      FILTER (STRSTARTS(STR(?val), "http://www.wikidata.org/value/"))
    }
  }
} AS %vals

# get all nodes
WITH {
  SELECT ?node WHERE {
    { SELECT (?item as ?node) WHERE { INCLUDE %items } }
    UNION
    { SELECT (?stmt as ?node) WHERE { INCLUDE %stmts } }
    UNION
    { SELECT (?ref as ?node) WHERE { INCLUDE %refs } }
    UNION
    { SELECT (?val as ?node) WHERE { INCLUDE %vals } }
  }
} as %nodes

WHERE {
  INCLUDE %nodes .
  ?node ?p ?o .
}
```

Listing A.1: SPARQL-Query für Person-Items (verwendet NamedSubqueries, eine SPARQL-Erweiterung von BlazeGraph)

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig, unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur angefertigt habe.

_____

Benno Fünfstück