Fachhochschule Graubünden
University of Applied Sciences

Fachhochschule Graubünden (FHGR)

Institut für Informationswissenschaften
Prof. Dr. habil. Wolfgang Semar

Knowledge Engineering and Extraction

Warhammer 40k Enzyklopädie

Erstellt von:
Mauro Stoffel
mauro.stoffel@stud.fhgr.ch

Studienrichtung:
MSc. Data Visualization

Dozent:
Albert Weichselbraun

Eingereicht am:
17.01.2024

Inhaltsverzeichnis

Abstract

Beschreibung

Warhammer 40000 (auch Warhammer 40k genannt) ist eine Fantasywelt, die, wie im Namen erwähnt, im 41. Jahrtausend stattfindet. Die Welt umfasst viele nennenswerte Charaktere verschiedener Rassen.
Das Ziel dieser Projektarbeit soll es sein, die im Internet verfügbaren Informationen zu sammeln und möglichst übersichtlich darzustellen, sodass Personen, welche sich neu in das Thema einlesen wollen, einen möglichst einfachen Start dafür haben.

Datensammlung

Eine erste Untersuchung hat gezeigt, dass es in WikiData bereits einige Einträge zum Thema gibt, welche jedoch noch Lücken aufweisen. Das Ziel der Datensammlung ist es nun also, die Daten des WikiData als Grundlage zu verwenden und mit Daten aus einem Fandom Wiki zu erweitern. Die Inhalte des Wikis sollen dabei über Named Entity Linking zu den bisherigen Daten hinzugefügt werden.

Datendarstellung

Die Daten sollen zum Schluss explorierbar gemacht werden. Die momentane Idee ist es über die Library Streamlit ein Dashboard zu erstellen, welches dem User ermöglichen soll, die gesammelten Daten nach verschiedenen Kategorien (Charaktere, Rassen, Ereignisse) zu untersuchen und so mehr über die Welt von Warhammer 40k zu erfahren.

Einleitung

Das folgende Projekt wurde im Rahmen der Projektarbeit des Fachs Knowledge Engineering and Extraction and FHGR entwickelt. Das Ziel war es dabei, Inhalte und Zusammenhänge rund um das Thema Warhammer 40k zu sammeln. Dabei handelt es sich um eine Fantasywelt, die in vielen verschiedenen Adaptionen, wie zum Beispiel Bücher, Videospiele und Tabletop-Games existiert. Es ist eine sehr umfangreiche und detaillierte Fantasywelt, welche eine Fülle von verschiedenen Rassen und Charakteren enthalten, wobei viele davon eine sehr detaillierte Hintergrundgeschichte haben. Um neue InteressentInnen einfacher in die Thematik einzuführen, wurden Daten aus verschiedenen Quellen zum Thema herangezogen, verarbeitet und in einem Dashboard dargestellt, welches es ermöglichen soll, die Daten frei zu explorieren. Sämtliche dabei entworfenen Inhalte und Codes sind auf dem folgenden Github-Link öffentlich zugänglich: (Link)

Knowledge Engineering

Datensammlung via WikiData

Zu Beginn der Datensammlung wurden die Inhalte von WikiData nach Informationen zu Warhammer 40k durchsucht. (Stand: 2.11.2024) Dabei wurde festgestellt, dass es drei relevante Hauptklassen gibt:

- Q209026 “Warhammer 40000” (*WikiData*, 2012)
- Q19595297 “Warhammer 40000 universe” (*WikiData*, 2015)
- Q928197 “Warhammer Fantasy” (*WikiData*, 2021)

Daraufhin wurden alle Klassen ausgegeben, die mit diesen Klassen verlinkt sind. Eine Analyse der Resultate hat gezeigt, dass es in den Inhalten der Hauptklassen Überschneidungen gibt und keine der drei Hauptklassen alle Resultate beinhaltet.

Klassen	1	2	3	1,2	1,3	2,3	1,2,3
Anzahl Resultat	20	34	32	53	35	65	68

Tabelle 1: Vergleich der Inhalte der Hauptklassen in WikiData

Somit wurden die Inhalte aller drei Hauptklassen mittels des UNION Befehl kombiniert. Die schematische Darstellung dieser Zusammenführung ist in Abbildung 1 dargestellt.

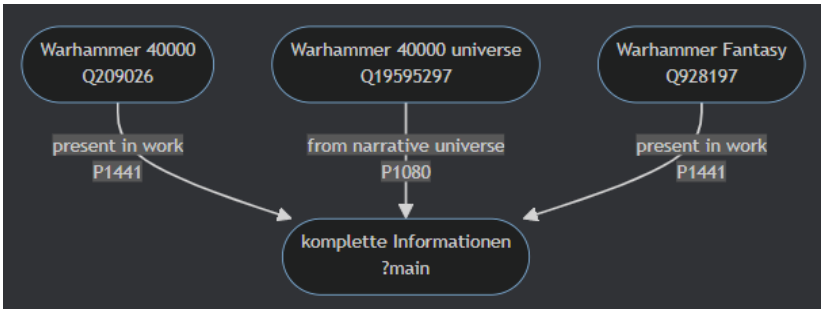

Abbildung 1: Schematische Darstellung der Zusammenführung der Basisinformationen von WikiData

Der Query wurde in Python mit der Library RDFLib (*RDFLib*, 2009) aufgerufen und die Resultate in einem Triple Store in der *base.ttl* Datei abgelegt. Der Code dafür wurde in Anhang 1: RDFLib Code Base Layer abgelegt.

Knowledge Extraction

Um die Daten mit mehr Informationen anzureichern, wurde als zweite Datenquelle das Fandom Wiki von Warhammer 40000 verwendet (*FandomWiki*, 2008). Dieses Wiki bietet für alle Personen mit einem Account die Funktion an, den gesamten Inhalt als Data-Dump im XML Format herunterzuladen.
Aus diesem Data-Dump wurden danach die einzelnen Pages, welche via page-tags unterteilt waren, extrahiert und in einem Dataframe mit den Spalten “title”, “id”, “link” und “text”. Die Inhalte der “link”-Spalte musste zuerst noch aus dem Titel der Page generiert werden, indem ein Basis-Link angehängt wurde und alle Leerzeichen im Titel mit einem Unterstrich ersetzt wurden.
Der Inhalt der “text”-Spalte ist mit dem Textinhalt der jeweiligen Seite gefüllt, wobei dieser mit der Wikisyntax codiert ist. Dieser Wikisyntax beinhaltet Funktionalitäten, um den Text nach einem standardisierten System zu formatieren (*Wikipedia*, 2004). Um so viele unterschiedliche Informationen wie möglich zu erhalten, wurde die Informations-Extraktion aus den Texten des Fandom Wikis auf 2 Arten durchgeführt.

1. Extraktion über den zuvor erwähnten Wikisyntax
2. Extraktion über Named-Entity-Recognition (NER) durch die Spacy-Library

Extraktion über Wikisyntax

Die Formatierung des Textes mit dem Wikisyntax bedeutet, dass sämtliche Inhalte der “text”-Spalte direkt via einer Regex-Suche nach Verlinkungen durchsucht werden können. Diese sind mittels doppelten eckigen Klammern gekennzeichnet. Die gefundenen Resultate können direkt als Named-Entities abgespeichert werden. Die erstellte Liste muss zum Schluss auf Duplikate durchsucht und mit den funktionierenden Links erweitert werden.

Extraktion über Named-Entity-Recognition

Für die Extraktion der Named Entities mit dem NER-Algorithmus von Spacy mussten die Seiten Texte zuerst gesäubert werden. Die Säuberungsschritte waren:

- Alle Worte kleinschreiben
- Alle Sonderzeichen entfernen

Die daraus entstandenen Texte wurden in einer neuen Spalte im Dataframe gespeichert, um sicherzustellen, dass der Originaltext noch erhalten bleibt.

Vergleich der Extraktionsmethoden

Vorteile Wikisyntax:

- schnell
- einheitliche Schreibweise → kaum Duplikate

- alle Named Entities haben direkt eine eigene Page, welche wiederum Referenzen enthält. So können sehr einfache Beziehungen aufgebaut werden

Nachteile Wikisyntax:

- eventuell weitere Named Entities welche nicht erkannt werden

Vorteile NER:

- sehr viele Named-Entities werden pro Seite erkannt
- gefundenen Named Entities werden direkt zusammen mit einem Typen Maker (engl. tag) abgelegt
- Es werden auch neue Typen von Named Entities erkannt (Locations, Zeiten…)

Nachteile NER

- sehr langsam
- unpräzise Erkennung der Named Entities → viele Duplikate mit unterschiedlichen Schreibweisen

Fazit

Da die Inhalte der Extraktion aus dem Wikisyntax qualitativ hochwertiger sind, werden sie für das weitere Vorgehen als Grundlage verwendet. Somit werden in den nächsten Schritte allen Named Entities, welche aus der Wikisyntax extrahiert wurden, passende Tags zugewiesen. Dafür gibt es 2 Priorisierungsstufen:

- Prio 1: Alle Named Entities, welche bereits in der *base.ttl* vorkommen, müssen nicht weiter untersucht werden, da sie bereits Tags besitzen
- Prio 2: Alle Named Entities, welche mit einem Element in der Liste der Named Entities aus der NER Extraktion übereinstimmen, übernehmen dieses Tag

Erweiterung des Triple Stores

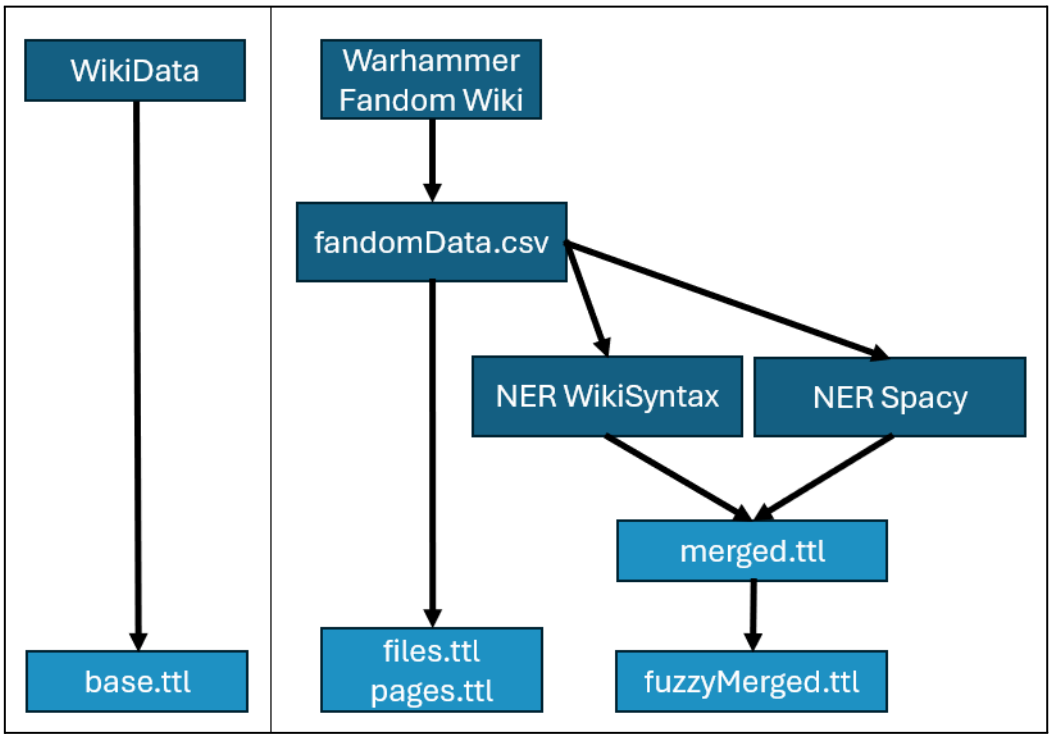

Abbildung 2: Herleitung der Erstellung der TripleStore Erweiterungen

Erstellung pages.ttl und files.ttl

Wie in Abbildung 2 gezeigt, konnten noch vor dem Beginn der NER Analysen direkt die Zusammenhänge zwischen gefundenen Named Entities und ihren Page-Verlinkungen und den Files, welche auf den Pages verwendet wurden, erstellt werden. Daraus entstanden die *pages.ttl* und *files.ttl* Dateien

Erstellung relations.ttl

Aus den Resultaten der Extraktion via WikiSyntax konnten die Zusammenhänge zwischen den einzelnen Named Entities aufgrund der Tatsache gemacht werden, dass sie jeweils auf derselben Page erwähnt wurden. Daraus entstand die *relations.ttl* Datei.

Erstellung des merged.ttl und fuzzyMerged.ttl

Im nächsten Schritt wurden die Daten aus der Named-Entity Liste mit allen Inhalten verglichen, die durch die SPARQL-Query extrahiert wurden. Da diese Named Entities bereits im Turtle-Store vorhanden sind, können sie ohne weitere Aktion aus der Liste gelöscht werden. Von den 17255 Einträgen blieben danach noch 17226 übrig.
Die restlichen Named Entities aus dem Wikisyntax wurden als nächstes mit den Named Entities verglichen, welche aus der Extraktion mit Spacy erstellt wurden. Daraus entstand die mergedEntityList.csv. Diese enthält noch 8260 Einträge, welche mit einem Spacy-Tag versehen sind. Um die Systeme aneinander anzugleichen, wurde für jeden Spacy-Tag eine

passende Klasse von WikiData definiert. Die Spacy-Tags und ihre dazugehörige WikiData Klasse sind in der Tabelle 2 dargestellt.

Spacy-Tag	WikiData Klasse
LOC	Location in a Fictional work: Q15796005
PERSON	Fictional Person: Q95074
NORP	Warhammer race: Q20667393
ORG	Fictional Organisation: Q14623646

Tabelle 2: Definition der WikiData Klassen für jedes gefundene Spacy-Tag

Nun konnten sämtliche Einträge als Turtle-Triplet dargestellt und in der neuen Datei *merged.ttl* serialisiert werden.
Zudem wurden alle neu abgelegten Entities ebenfalls aus der originalen Liste der Named entities entfernt. Da nach dem Entfernen der gefundenen Entities noch immer 8967 Einträge in der Liste der gefundenen Named Entities übrig waren, wurde die Suche noch erweitert. Es wurden mit einem Fuzzy Filter die Namen der Named Entities der WikiData Liste und die Namen der Spacy Liste verglichen. Wenn eine Übereinstimmung von mehr als 80% gefunden wurde, wurden die Daten ebenfalls übernommen. Mit dieser Methode konnten nochmals knapp 2500 Einträge gefunden werden. Diese Einträge wurden in der *fuzzyMerged.ttl* Datei abgelegt.

Alle diese separaten Triple Stores also das *base.ttl, files.ttl, pages.ttl, merged.ttl* und *fuzzyMerged.ttl* wurden zu einem Triple Store kombiniert, welcher die Grundlage für den im nächsten Kapitel besprochenen Reasoning Prozess war.

Ontologie

Die Ontologie wurde dafür verwendet, die Informationen aus den verschiedenen Datenquellen zu vereinheitlichen.
Die Informationen, welche aus WikiData extrahiert wurden, bestehen aus vielen verschiedenen Klassen. Um die Handhabung in der Visualisierung zu vereinfachen, wurden für die vier Hauptgruppen Charakter, Rasse, Organisation und Location eine Überklasse definiert und alle anderen Klassen mit der rdfs:subClassOf Property ihrer jeweiligen Überklasse zugewiesen. Die daraus resultierende Ontologie ist in der Abbildung 3 zu sehen.

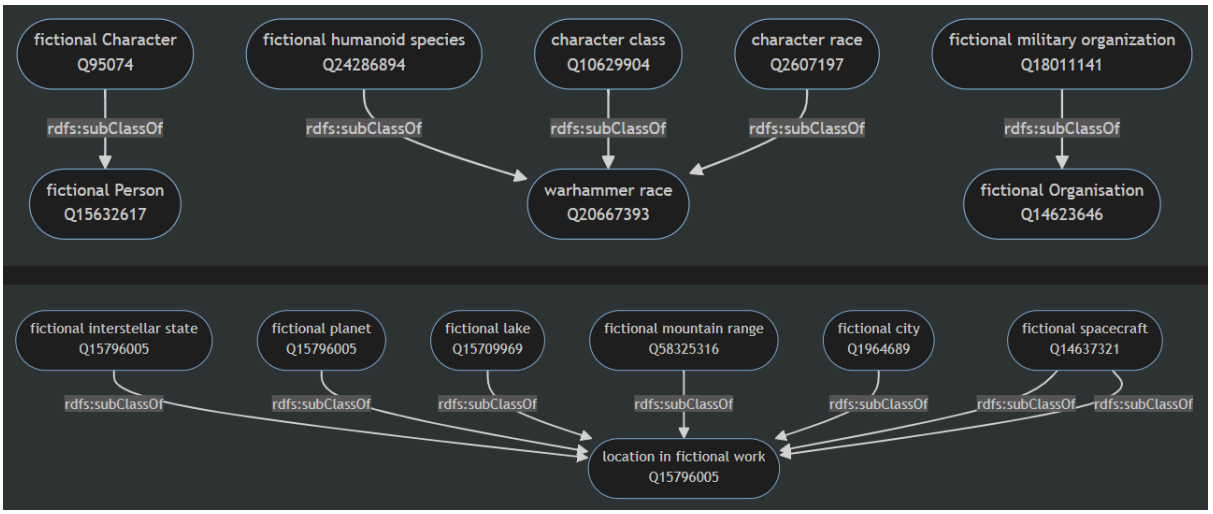

Abbildung 3: Schema Ontologie

Nachdem die Ontologie definiert war, wurde sie mit dem Reasoner aus dem OWL-RL Paket (*OWL-RL*, 2018) auf die in den vorherigen Kapiteln zusammengetragenen Daten angewendet. Daraus entstand der Triple Store, welcher im nächsten Kapitel für die Visualisierung verwendet wurde.

Visualisierung

Für die Visualisierung wurden die Programmiersprache Python und die Streamlit Bibliothek (*Streamlit*, 2019) verwendet.
Beim Aufstarten einer neuen Instanz werden die Daten aus dem Triple Store eingelesen und in ihre Klassen unterteilt. Dies wird gemacht, um die Filterlisten zu füllen, welche in der Abbildung 4 auf der linken Seite zu sehen sind. Diese Filterlisten ermöglichen dem User nach spezifischen Informationen zu suchen, die für sie oder ihn interessant sind. Nach Auswahl einer Option werden das Bild der Auswahl, der Link zu den Hintergrunddaten im Fandom Wiki und ein RDF Graph dargestellt.
Der RDF Graph zeigt alle Beziehungen der ausgewählten Option zu anderen Named Entities, welche im Triple Store existieren. Der User kann nun durch den Triple Store navigieren, indem er oder sie entweder eine neue Suche startet, oder eine der gefundenen Beziehungen im RDF Graph auswählt, um diese zu öffnen.

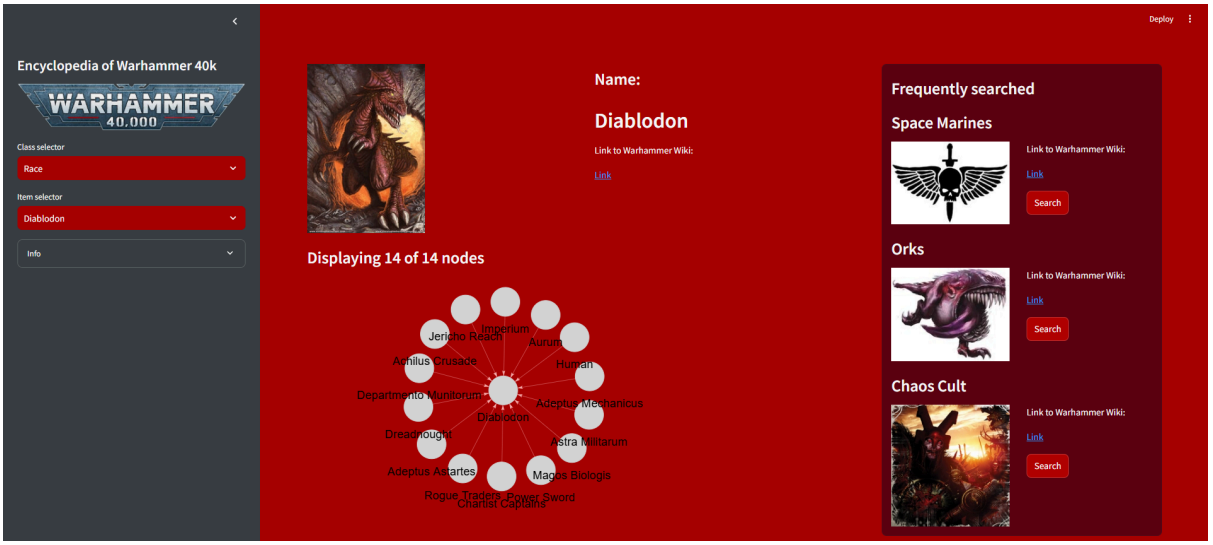

Abbildung 4: Visualisierung mit Streamlit

Auch wurde es dem User ermöglicht, auf die häufigsten Suchbegriffe direkt zuzugreifen. Diese sind auf der rechten Seite des Dashboards zu sehen.

Fazit und Limitationen

Die verwendete Kombination der Prozesse zur Findung von Named Entities über die Wiki-Syntax und über den NER-Algorithmus von Spacy hat sich als effektive Methode zur Erstellung des Triple Stores gezeigt. Dies kann für zukünftige Projekte gut wieder adaptiert werden.
Die Visualisierung erlaubt es dem User sehr intuitiv die Daten zu durchsuchen und neue Einblicke und Zusammenhänge zu gewinnen. Ein Problem hier ist die Darstellung der Nodes, da bei zu vielen Beziehungen die Übersichtlichkeit sehr schnell abnimmt. Daher musste auch eine Begrenzung eingebaut werden, um dies zu vermeiden. In einem Folgeprojekt wäre es hier wahrscheinlich sinnvoller, eine tabellarische Darstellung zu verwenden.
Eine weitere Limitation ist die Art der Beziehungen. Diese bestehen im Moment nur aus dem Zusammenhang, dass die Elemente auf derselben Seite im Fandom-Wiki vorkommen. Eine Erweiterung welche die Art der Beziehungen besser beschreibt, wie zum Beispiel “istRasse” wäre sehr mächtig und könnte weitere neue Erkenntnisse erzeugen.
Trotzdem erlaubt das erstellte Tool die Informationen auf eine neue Weise zu durchsuchen und ermöglicht dem User, sich in die verschiedenen Themengebiete einzuarbeiten.

Quellenverzeichnis

FandomWiki. (2008, May 12). Warhammer 40k Wiki Warhammer 40k Wiki. Retrieved January 12, 2025, from https://warhammer40k.fandom.com/wiki/Warhammer_40k_Wiki

OWL-RL. (2018, November 23). OWL-RL. https://owl-rl.readthedocs.io/en/latest/#

RDFLib. (2009). rdflib 7.1.2 — rdflib 7.1.2 documentation. Retrieved December 12, 2025, from https://rdflib.readthedocs.io/en/7.1.1/index.html

Streamlit. (2019, April 26). Streamlit • A faster way to build and share data apps. Retrieved December 11, 2025, from https://streamlit.io/

WikiData. (2012, November 30). WikiData Warhammer 40000. Retrieved 11 2, 2024, from https://www.wikidata.org/wiki/Q209026

WikiData. (2015, March 15). WikiData Warhammer 40000 universe. Retrieved 11 2, 2024, from https://www.wikidata.org/wiki/Q19595297

WikiData. (2021, June 15). WikiData Warhammer Fantasy. Retrieved 11 2, 2024, from https://www.wikidata.org/wiki/Q15831558

Wikipedia. (2004, June 14). Wikipedia Hilfe:Textgestaltung. https://de.wikipedia.org/wiki/Hilfe:Textgestaltung

Anhang

Anhang 1: RDFLib Code Base Layer

```
import pandas as pd
from rdflib import Graph, URIRef, Namespace
from rdflib.namespace import RDFS

def getBaseData():
   QUERY = '''
   PREFIX wdt: <http://www.wikidata.org/prop/direct/>
   PREFIX wd: <http://www.wikidata.org/entity/>
   PREFIX wikibase: <http://wikiba.se/ontology#>
   PREFIX bd: <http://www.bigdata.com/rdf#>
   SELECT DISTINCT ?main ?mainLabel ?type ?typeLabel
   WHERE{
       SERVICE <https://query.wikidata.org/sparql>{
         # 1 present in work Warhammer 40000 == Viele verschiedene Outputs
         {?main wdt:P1441 wd:Q209026.}
         UNION
         # 2 present in work Warhammer == Viele verschiedene Outputs
         {?main wdt:P1441 wd:Q928197.}
         UNION
         # 3 from_narrative universe == Viele verschiedene Outputs
         {?main wdt:P1080 wd:Q19595297.}

         ?main wdt:P31 ?type.
         ?main rdfs:label ?mainLabel filter (lang(?mainLabel) = "en").
         ?type rdfs:label ?typeLabel filter (lang(?typeLabel) = "en").
       }

       SERVICE wikibase:label { bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],en". }}
   '''

   g = Graph()

   WDT = Namespace("http://www.wikidata.org/prop/direct/")
   g.bind("wdt", WDT)

   for main, mainLabel, type, typeLabel in g.query(QUERY):
       print(main, mainLabel, type, typeLabel)
       g.add((main, WDT.P31, type))
       g.add((main, RDFS.label, mainLabel))
       g.add((type, RDFS.label, typeLabel))


   g.serialize('../dok/base.ttl', format='ttl')

getBaseData()
```