<u>Process to extract WikiData (English) and migrate it to Hindi WikiData recursively</u>

Algorithm will be like

1.Create a crawler which will get list of all important articles added recently and in past few years on English Wikidata we will assign importance to these articles by considering selective factors

2.Once we have some data in source connect to wikiData and create retrieve all details of given article one at a time and store it in DB (It will be a NO SQL DB as we have media to store as well)

3.use some open source lang translator to translate and store data under specific section like title , history .. etc.

3.Use a simple JavaScript / python script to create a article automatically from DB data on target Wiki (Hindi in our case)

4. Mark the flag as true (article created in destination) in source file or remove the entry of article which is created in destination

5.Link the newly created articles to other language if that article is already present in some other language

5.repeat 2 to 5 recursively until all source data migrated to destination

6. crawling is also a recursive operation either scheduled in timely manner or manual trigger

1.extract wikiData (English) and store it in DB which can be used for creating simillar pages in Hindi or other languare

-we can use any open src python tool to interact with wikiData

-say we are using qwikidata

-qwikidata is a Python package with tools that allow you to interact with Wikidata.

The package defines a set of classes that allow you to represent Wikidata entities in a Pythonic way. It also provides a Pythonic way to access three data sources,

linked data interface

sparql query service

json dump

Quick Install

Requirements

- python >= 3.5

Install with pip

- You can install the most recent version using pip,

```
pip install qwikidata
```

Linked Data Interface

```python
from qwikidata.entity import WikidataItem, WikidataLexeme, WikidataProperty
from qwikidata.linked_data_interface import get_entity_dict_from_api

# create an item representing "Douglas Adams"
Q_DOUGLAS_ADAMS = "Q42"
q42_dict = get_entity_dict_from_api(Q_DOUGLAS_ADAMS)
q42 = WikidataItem(q42_dict)

# create a property representing "subclass of"
P_SUBCLASS_OF = "P279"
p279_dict = get_entity_dict_from_api(P_SUBCLASS_OF)
p279 = WikidataProperty(p279_dict)

# create a lexeme representing "bank"
L_BANK = "L3354"
l3354_dict = get_entity_dict_from_api(L_BANK)
l3354 = WikidataLexeme(l3354_dict)
```

Use SPARQL Query Service to get data (we will loop this to get data recursively recursively)

This will pick list of queries from a input source and keep on extracting data from English wikidata and store in DB for further process and filter list of doc to be retrieved accordingly and we will also keep on updating this source list for the newly added articles in English wiki

```python
from qwikidata.sparql import (get_subclasses_of_item,
                              return_sparql_query_results)

# send any sparql query to the wikidata query service and get full result back
# here we use an example that counts the number of humans
sparql_query = """
SELECT (COUNT(?item) AS ?count)
WHERE {
        ?item wdt:P31/wdt:P279* wd:Q5 .
}
"""
res = return_sparql_query_results(sparql_query)


# use convenience function to get subclasses of an item as a list of item ids
Q_RIVER = "Q4022"
subclasses_of_river = get_subclasses_of_item(Q_RIVER)
```

JSON Dump

```python
import time

from qwikidata.entity import WikidataItem
from qwikidata.json_dump import WikidataJsonDump
from qwikidata.utils import dump_entities_to_json

P_OCCUPATION = "P106"
Q_POLITICIAN = "Q82955"

def has_occupation_politician(item: WikidataItem, truthy: bool = True) -> bool:
    """Return True if the Wikidata Item has occupation politician."""
    if truthy:
        claim_group = item.get_truthy_claim_group(P_OCCUPATION)
    else:
        claim_group = item.get_claim_group(P_OCCUPATION)

    occupation_qids = [
        claim.mainsnak.datavalue.value["id"]
        for claim in claim_group
        if claim.mainsnak.snaktype == "value"
    ]
    return Q_POLITICIAN in occupation_qids

# create an instance of WikidataJsonDump
wjd_dump_path = "wikidata-20190401-all.json.bz2"
wjd = WikidataJsonDump(wjd_dump_path)

# create an iterable of WikidataItem representing politicians
politicians = []
t1 = time.time()
for ii, entity_dict in enumerate(wjd):

    if entity_dict["type"] == "item":
        entity = WikidataItem(entity_dict)
        if has_occupation_politician(entity):
            politicians.append(entity)

    if ii % 1000 == 0:
        t2 = time.time()
        dt = t2 - t1
        print(
            "found {} politicians among {} entities [entities/s: {:.2f}]".format(
                len(politicians), ii, ii / dt
            )
        )
    if ii > 10000:
        break
# write the iterable of WikidataItem to disk as JSON
out_fname = "filtered_entities.json"
dump_entities_to_json(politicians, out_fname)
wjd_filtered = WikidataJsonDump(out_fname)

# load filtered entities and create instances of WikidataItem
for ii, entity_dict in enumerate(wjd_filtered):
    item = WikidataItem(entity_dict)
```

once we got Jason dump for data which we looking for we will use a simple python / JavaScript script to update these details articlewise in hindi wiki and linked it with all available sources

As this is a recursive algo it will run until all source documents gets migrated to selected language

Github Link : https://github.com/Arjungambhir/WikiBotForHindi (not yet pushed any code in repo yet just started with basic papers and algorithm once done with dev will push it to same repo)

Wiki Link :
"https://hi.wikipedia.org/wiki/%E0%A4%AC%E0%A4%B0%E0%A5%8D%E0%A4%B2%E0%A4%BF%E0%A4%A8_%E0%A4%95%E0%A5%80_%E0%A4%A6%E0%A5%80%E0%A4%B5%E0%A4%BE%E0%A4%B0_%E0%A4%95%E0%A4%BE_%E0%A4%97%E0%A4%BF%E0%A4%B0%E0%A4%A8%E0%A4%BE"

Note : this article is created manually with ref to English article and linked to all other available language once algorithm mentioned above implemented it will be a automated process

Some Basic Ref and Concepts

How to create Wikipedia pages from Wiki Data?

Wikidata acts as central storage for the structured data of its Wikimedia sister projects including Wikipedia (...).

Loosely, we could describe Wikidata as Wikipedias database with over 46million data items.

And in line with Wikimedia's mission, everyone can add and edit data, and use it for free.

Available data

Like Wikipedia, there are all kinds of data stored in Wikidata. As such, when you are looking for a specific dataset or if you want to answer a curious question, it can be a good start looking for that data at Wikidata first

Advantages and Disadvantages of Wikidata

There are some aspects you should keep in mind when using Wikidata. Whether they are an advantage or disadvantage, however, depends on you:

- a free and open knowledge base that can be read and edited by both humans and machines
- contains various data types (e.g. text, images, quantities, coordinates, geographic shapes, dates)
- uses SPARQL

Especially the last aspect allows you very interesting questions like to ones above. If you have never used SPARQL before, however, it might be a struggle in the beginning. But don't worry. The next section gives you a brief introduction.
Wikidata is a Wikimedia project to create an open and collaborative database. It stores relational statements about an entity as well as the interwiki links associated with the pages on the Wikimedia projects that describe that entity. The English Wikipedia uses these interlanguage links stored at Wikidata, and has some limited applications for the statements made in Wikidata.

Interlanguage links

Each Wikipedia page with an entry in Wikidata uses the language links stored there to populate the language links that show in the left column. Traditional interwiki links in a page's wiki-text are still recognized, and simply override the information for that language (if any) from Wikidata.

If the article is linked to from Wikidata, then it will display all links listed there, in addition to any links entered in the article wiki-text. In the case of a conflict, or intentional difference, between a local link and a Wikidata link for a given language, the local link will be displayed; the Wikidata links for all other languages will still display.

If the article is not yet linked to from Wikidata, then it will display whatever links are in the article wiki-text, as before. In some cases, there may be a Wikidata item that corresponds to the article, but the English Wikipedia article has not yet been associated with it. In the course of routine maintenance at Wikidata, these instances will be fixed.

Refer the article created for "बर्लिन की दीवार का गिरना" in hindi and linked to other languages

Access WIkidata

The first way to access data is to use the *#statements* parser function. This function will allow you to display the value of any statement included in an item.

Direct access

On a page that is connected to a Wikidata item via the interwiki links, you can use the function by adding the label of the property you want in your language or the P-number of the property. The code has to be added in the wikicode.

Examples:

- `{{#statements:member of political party}}` or `{{#statements:P102}}` will return the "member of political party" value.
- `{{#statements:discoverer or inventor}}` or `{{#statements:P61}}` will return the "discoverer or inventor" value.
- On w:en:Douglas Adams, the code `{{#statements:country of citizenship}}` will display "United Kingdom".

Arbitrary access

You can also display data from an item that is not connected via an interwiki link. For this, you use the same function, adding a parameter from= followed by the Q-id of the item.

Examples:

- `{{#statements:birth name|from=Q42}}` will display "Douglas Noël Adams"
- `{{#statements:country of citizenship|from=Q42}}` will display "United Kingdom".
- `{{#statements:P1476|from=Q191380}}` will display "Notre-Dame de Paris"
- `{{#statements:author|from=Q191380}}` will display "Victor Hugo"
- `{{#statements:publication date|from=Q191380}}` will display "14 January 1831"

Multiple values

When a statements has multiple values, the parser function will show the "best" value, which means:

- only show the preferred value(s) if there are any
- if not, shows all the values
- but never the deprecated ones

Example: `{{#statements:occupation|from=Q42}}` displays

"playwright, screenwriter, novelist, children's writer, science fiction writer, comedian" (there are other occupations in Douglas Adams (Q42) but only some are preferred)

Formatted values

For some properties, the #statements parser function will display the value in a specific format.

Geo coordinates

The parser function displays the coordinates in degree-minute-second format.

`{{#statements:coordinate location|from=Q243}}` displays 48°51′30″N 2°17′40″E

Monolingual text

The parser function displays the string, annotated with the language.

`{{#statements:motto text|from=Q95}}` displays "`Do the right thing`".

Date

The date value will be formatted in day-month-year format.

`{{#statements:date of birth|from=Q42}}` displays `11 March 1952`

Links

The links are clickable.

`{{#statements:official website|from=Q243}}` displays http://www.toureiffel.paris, https://www.toureiffel.paris/en

External IDs

An external ID will provide a direct link to the external website.

`{{#statements:IMDb ID|from=Q42}}` displays nm0010930

Raw value

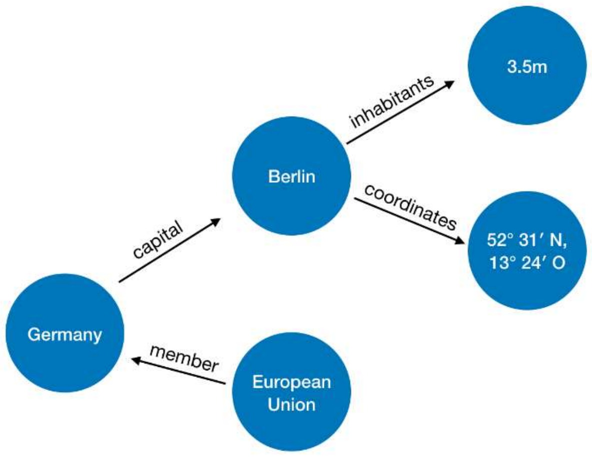
To display the unlinked value, use `#property`.

Sample:

`{{#property:IMDb ID|from=Q42}}` displays `nm0010930`

Idea and Concept of SPARQL

SPARQL is a query language for RDF databases. In contrast to relational databases like SQL, items are not part of any tables. Instead, items are linked with each other like a graph or network:



To describe these relations, we can use a triple:

A triple is a statement containing a subject predicate and object.

How to query data from Wikidata?

SPARQL is a language to formulate questions (queries) for knowledge databases. With the right database, a SPARQL query could answer questions like “what is the most popular tonality in music?” or “which character was portrayed by the most actors?” or “what’s the distribution of blood types?” or “which authors’ works entered the public domain this year?”.

To get data from Wikidata you simply use triples (like to one above) to write a SPARQL query. Let’s have a look how such a SPARQL query might look like. Note, that we are using specific identifiers to define the right relationship and item:

```
SELECT ?country
WHERE
{
  ?country   wdt:P463     wd:Q458.  #country   #member of
#European Union
}
```

Here, we simply ask for the countries that are part of the European Union.

Do you recognize the subject-predicate-object statement? We just select those countries, for which the condition holds: the country (?country) is a member of (wdt:P463) the European Union (wd:Q458).

Expressions, FILTER and BIND

This section might seem a bit less organized than the other ones, because it covers a fairly wide and diverse topic. The basic concept is that we would like to do something with the values that, so far, we've just selected and returned indiscriminately. And expressions are the way to express these operations on values. There are many kinds of expressions, and a lot of things you can do with them – but first, let's start with the basics: data types.

Data types

Each value in SPARQL has a type, which tells you what kind of value it is and what you can do with it. The most important types are:

- item, like `wd:Q42` for Douglas Adams (Q42).
- boolean, with the two possible values `true` and `false`. Boolean values aren't stored in statements, but many expressions return a boolean value, e.g. `2 < 3` (`true`) or `"a" = "b"` (`false`).
- string, a piece of text. String literals are written in double quotes.
- monolingual text, a string with a language tag attached. In a literal, you can add the language tag after the string with an `@` sign, e.g. `"Douglas Adams"@en`.
- numbers, either integers (`1`) or decimals (`1.23`).
- dates. Date literals can be written by adding `^^xsd:dateTime` (case sensitive – `^^xsd:datetime` won't work!) to an ISO 8601 date string: `"2012-10-29"^^xsd:dateTime`.

Operators

The familiar mathematical operators are available: `+`, `-`, `*`, `/` to add, subtract, multiply or divide numbers, `<`, `>`, `=`, `<=`, `>=` to compare them. The inequality test ≠ is written `!=`. Comparison is also defined for other types; for example, `"abc" < "abd"` is true (lexical comparison), as is `"2016-01-01"^^xsd:dateTime > "2015-12-31"^^xsd:dateTime` and `wd:Q4653 != wd:Q283111`. And boolean conditions can be combined with `&&` (logical and: `a && b` is true if both `a` and `b` are true) and `||` (logical or: `a || b` is true if either (or both) of `a` and `b` is true).

FILTER

`FILTER(condition).` is a clause you can insert into your SPARQL query to, well, filter the results. Inside the parentheses, you can put any expression of boolean type, and only those results where the expression returns `true` are used.