qapedia

Versão 0.1.0

03 jun, 2019

Documentação

O QApedia é uma ferramenta de geração de pares de questão-sparql escrito em Python a partir do endpoint previamente estabelecido. O seu nome vem da junção dos termos QA de *Question-Answering* e *pedia* da palavra wikipedia.

CAPÍTULO 1

Introdução

1.1 O que é o QApedia?

QApedia («QA» de Question-Answering, «pedia» de Wikipédia) é um pacote escrito em python que tem como objetivo realizar a geração de pares de questões-queries com base em um template previamente estabelecido sobre o endpoint da DBpedia. Esses pares gerados podem ser utilizado na criação de um sistema de *Question-Answering (QA)*, onde a pergunta é fornecida em linguagem natural e a sua resposta seria o resultado da consulta Sparql sobre a base de dados da DBpedia.

1.1.1 DBpedia

A **DBpedia** («DB» de database) é um projeto criado pela comunidade que cria e fornece acesso aos dados estruturados extraídos da Wikipédia. Esses dados possuem conexões de modo que consultas possam ser realizadas em cima da Wikipédia similarmente a consultas realizadas em banco de dados, esses dados também são conhecidos como *Linked Open Data*. Tim Berners-Lee listou quatro princípios da *Linked Data* em sua nota Linked Data, são eles:

1. Use Uniform Resource Identifier (URIs) para nomear para coisas
2. Use HTTP URIs para que as pessoas possam procurar esses nomes.
3. Quando alguém procura um URI, forneça informações úteis, usando os padrões (RDF*, SPARQL)
4. Inclua links para outros URIs. Para que eles possam descobrir mais coisas.

Na imagem a seguir é mostrada a página que representa uma entidade do tipo evento, a entidade no caso é Balaiada que possui uma identificação URI, além de um conjunto de informações agregadas, como o local onde ocorreu esse evento (`dbo:place`) ou um breve resumo sobre esse evento (`dbo:abstract`).

About: Balaiada

An Entity of Type : event, from Named Graph : http://dbpedia.org, within Data Space : dbpedia.org

The Balaiada was a social revolt that occurred between 1838 and 1841 in the interior of the province of Maranhão, Brazil.

Property	Value
dbo:abstract	■ The Balaiada was a social revolt that occurred between 1838 and 1841 in the interior of the province of Maranhão, Brazil. (en)
dbo:combatant	■ Rebels ■ *15pxNational Guard ■ *15pxImperial Navy ■ Empire of Brazil ■ *African slaves ■ *Balaios
dbo:commander	■ dbr:Luís_Alves_de_Lima_e_Silva ■ dbr:Cosme_Bento ■ dbr:Raimundo_Gomes
dbo:place	■ dbr:Brazil
dbo:result	■ Legalist victory
dbo:strength	■ 8,000 ■ 10,000-12,000
dbo:thumbnail	■ wiki-commons:Special:FilePath/Tropas_brasileiras_1835.jpg?width=300
dbo:wikiPageID	■ 12900447 (xsd:integer)
dbo:wikiPageRevisionID	■ 733591416 (xsd:integer)
dbp:caption	■ Troops departing from Rio de Janeiro to Maranhão.
dbp:date	■ 1838 (xsd:integer)
dct:subject	■ dbc:Maranhão ■ dbc:19th-century_conflicts ■ dbc:Rebellions_in_Brazil

1.1.2 Dados RDF

O Resource Description Framework (RDF) é um framework que serve para expressar informações sobre recursos. Qualquer coisa pode ser considerada um recurso, seja ela um conceito abstrato, documentos, pessoas ou objetos físicos, por exemplo. O RDF permite realizar declarações sobre os recursos sempre seguindo a seguinte estrutura:

<sujeito><predicado><objeto>

Através de uma instrução RDF, os recursos (sujeito e objeto) são relacionados através de um predicado que representa a natureza desse relacionamento. Sendo esse relacionamento direcional (do sujeito para o objeto) nomeado propriedade em RDF. A seguir, um exemplo de triplas RDF representado em pseudocode do recurso **Balaiada**.

<Balaiada> <é um> <evento>.
<Balaiada> <é a história das> <Forças Armadas Brasileiras>.
<Balaiada> <foi em> <1838>.

<Balaiada> <foi no> <Brasil>.
<Balaiada> <teve como resultado> <vitória legalista>

O conjunto de dados RDF da DBpedia é hospedado e publicado usando o OpenLink Virtuoso. Através da infraestrutura fornecida pelo Virtuoso é possível acessar os dados RDF da DBpedia através do endpoint SPARQL usando GET padrão de cliente WEB HTTP. Esse endpoint é utilizado na realização de buscas SPARQL realizadas pelo pacote `qapedia` para recuperar esses dados da Wikipedia.

CAPÍTULO 2

Tutorial

2.1 Introdução

Necessita de um conjunto de questões-sparql? Com o *QApedia* vamos mostrar como realizar essa tarefa de forma a ajudá-lo nesse problema de construção desse dataset.

Este tutorial explica o processo de geração dos pares de questão-sparql utilizando o pacote *QApedia*, desde o formato do arquivo de entrada utilizado até o resultado gerado. Primeiramente vamos definir algumas informações importantes utilizadas nesse documento.

- **SPARQL Endpoint** - URL onde é possível realizar consultas SPARQL com resultados via HTTP.
- **question** (pergunta) - pergunta em linguagem natural
- **query** - resposta da pergunta em formato de SPARQL
- **lacuna** (placeholder) - campo presente tanto na *pergunta* quanto na *query* configurando uma pergunta/query genérica.
- **generator query** (query geradora) - *query* que gera as possíveis opções que podem ser utilizadas para preencher as *lacunas*.
- **template** - Estrutura contendo os elementos *pergunta*, *query* e *generator query*
- **rdfs:label** - usada para fornecer uma versão legível do nome de um recurso.
- **URI** - Identificador de um recurso abstrato ou físico. Exemplos:

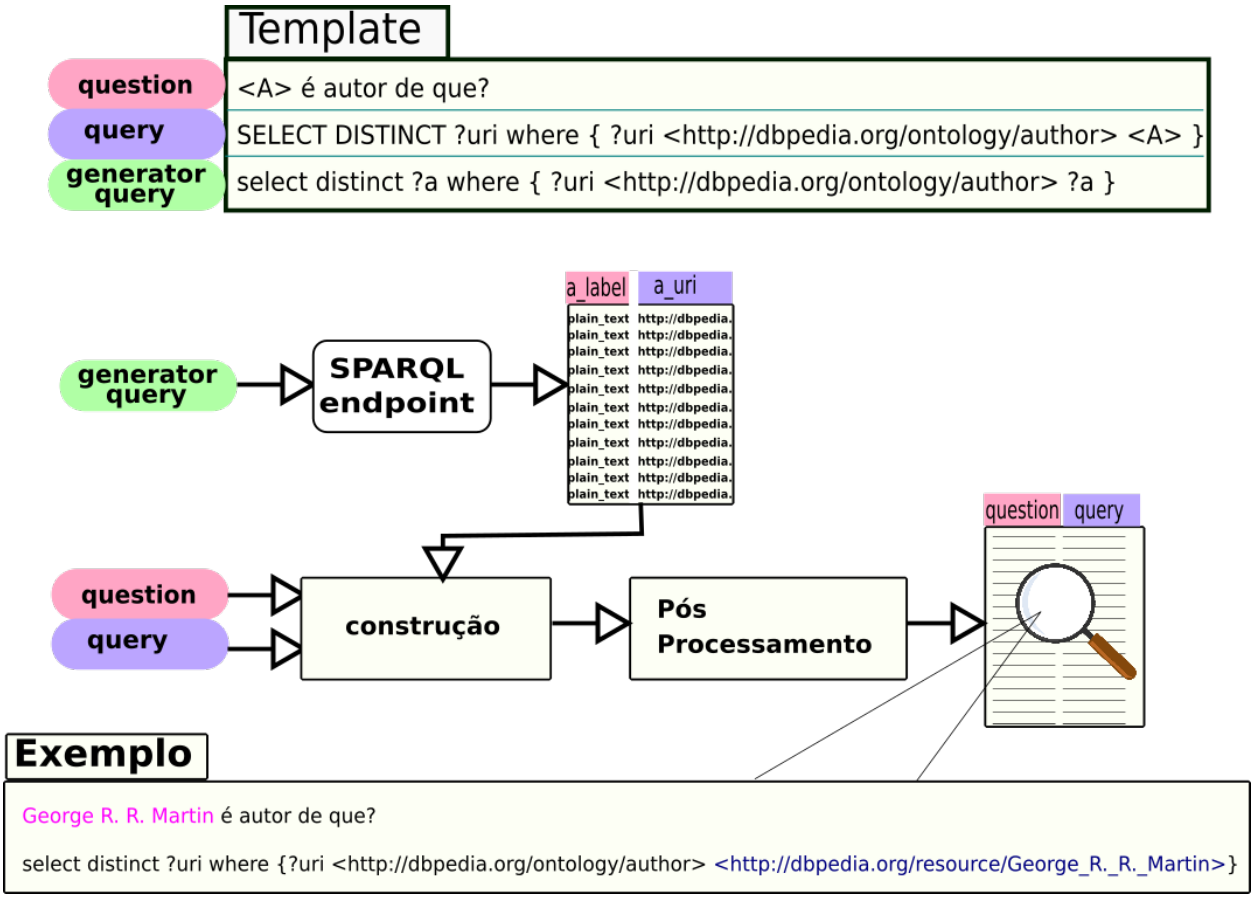
```
http://dbpedia.org/resource/Hunter_%C3%97_Hunter
https://pt.wikipedia.org/wiki/Hypertext_Transfer_Protocol
https://qapedia.readthedocs.io/pt/latest/
```

- **Prefix** - São abreviações de URIs. São similares os *imports* e *includes*, onde é definida as abreviações utilizadas.
- **resource** (recurso) - são representados com URIs, e podem ser abreviados como nomes prefixados.

Uma consulta SPARQL geralmente pode ser representada da seguinte forma.

```
# prefix declarations
# nomeia uma URI de forma a abreviar
PREFIX foo: <http://example.com/resources/>
...
# dataset definition
# Indica os grafos RDFS consultados
FROM ...
# result clause
# Indica a informação a ser retornada da consulta
SELECT ...
# query pattern
# Especifica o que consultar no conjunto de dados
WHERE {
    ...
}
# query modifiers
# reorganiza os dados da consulta
# pode limitar a quantidade de resultados, reordenar, etc.
ORDER BY ...
```

Na Figura abaixo temos o processo de geração de um conjunto de pares de questão-sparql, onde desejamos obter o formato de *pergunta* «Fulano é autor de que?» e a *query* «select ?obra where {?obra tem_autor Fulano}». O *placeholder* é definido no formato `<LETRA>` e está presente na *question* e *query*.



A *generator_query* é utilizada para realizar uma consulta no endpoint especificado e ajustada para retornar os campos de *rdfs:label* e a *URI* do recurso autor. Com esse resultado, preenchemos as lacunas presente no template e geramos pares similares ao do exemplo mostrado. Nas próxima seção é mostrado o guia de usuário contendo instruções de uso

das funções do pacote.

CAPÍTULO 3

Guia de Usuário

3.1 Instalação

A priori, para se utilizar o pacote do QApedia é necessário realizar a instalação do mesmo. Ela pode ser executada através de dois modos:

3.1.1 Através das releases

Em Releases o download do código pode ser realizado através do zip ou tar.gz. No terminal, para realizar a instalação, pode ser executar as seguintes instruções:

```
foo@bar:~$ wget https://github.com/JessicaSousa/QApedia/archive/0.1.0.tar.gz
foo@bar:~$ tar -xvzf 0.1.0.tar.gz
foo@bar:~$ cd QApedia-0.1.0/
foo@bar:~/QApedia-0.1.0$ pip install .
```

3.1.2 Através do github

Você pode pode realizar git clone no repositório e instalar usando os comandos a seguir

```
foo@bar:~$ git clone https://github.com/JessicaSousa/QApedia.git
foo@bar:~$ cd QApedia
foo@bar:~/QApedia$ pip install .
```

3.2 Carregando os templates

Carregue o pacote para poder utilizar suas funcionalidades

```
>>> import QApedia
```

Você pode carregar o seu dataset utilizando a função *QApedia.io.load_templates()* presente no pacote. O arquivo do dataset deve estar no formato csv, a primeira linha deve conter o nome das colunas. Esses nomes devem ser os mesmos que são mostrados no *Tutorial*. São eles:

- **query** - consulta SPARQL contendo uma lacuna no formato `<LETTER>`.
- **question** - pergunta em linguagem natural contendo a mesma lacuna presente na *query*.
- **generator_query** - query utilizada para preencher as lacunas e assim permitir a geração do conjunto de `question-query`.

Na tabela seguir é mostrado um exemplo de dataset contendo apenas um template.

question	query	generator_query
<A> é autor de que?	select distinct ?uri where { ?uri dbo:author <A>}	select distinct ?a where { ?uri dbo:author ?a}

O template é carregado como um `pandas.Dataframe`, então as operações de Dataframe podem ser realizadas em cima do conjunto de dados. As variáveis presentes no select da **generator_query** são extraídas no processo de leitura do conjunto de dados e são disponibilizadas na coluna **variables**.

```
>>> from QApedia import io
>>> templates = io.load_templates("templates.csv")
>>> templates.head()
                                        question  ... variables
0        <A> e <B> é produzido por qual empresa?  ...    [a, b]
1  <A> e <B> é o trabalho notável de qual autor?  ...    [a, b]
2         <A> e <B> são escritos por qual autor?  ...    [a, b]
3                       <A> escreveu qual livro?  ...       [a]
4          <A> pertence a qual partido político?  ...       [a]

[5 rows x 4 columns]
```

Para o exemplo mostrado na tabela anterior, sobre a **generator_query** «*select distinct* **?a** *where { ?uri dbo:author ?a}*», a variável extraída seria o *a*.

3.3 Ajustando a *query geradora*

Uma *generator_query* possui o seguinte formato:

```
#<A> e <B> são os trabalhos notáveis de qual escritor?
select distinct ?a ?b where {
    ?uri <http://dbpedia.org/property/notableworks> ?a .
    ?uri <http://dbpedia.org/property/notableworks> ?b .
    ?uri a <http://dbpedia.org/ontology/Writer>
}
```

Onde *a* e *b* correspondem as lacunas *<A>* e *<B>* definidas na *questão* e *query* do respectivo template. Nesse exemplo, a consulta irá retornar uma lista contendo os URIS dos trabalhos notáveis *a* e *b* de cada escritor. Entretanto, para preencher a lacuna da pergunta em linguagem natural, necessitamos do nome desses recursos em linguagem natural, para isso, são extraídas a propriedade **rdfs:label** de cada uma dessas URIs. As *generator_query* definida

em nosso template não possui esses campos, então para isso pode ser utilizada a função *QApedia.generator.adjust_generator_query()* que irá inserir as variáveis `la` e `lb` que correspondem as *labels* extraídas sobre cada recurso.

```
>>> from QApedia import generator
>>> generator_query = "select distinct ?a ?b where {"\
...                   "?uri <http://dbpedia.org/property/notableworks> ?a . "\
...                   "?uri <http://dbpedia.org/property/notableworks> ?b . "\
...                   "?uri a <http://dbpedia.org/ontology/Writer> }"
>>> variables = ["a", "b"]
"select distinct ?a ?b ?la ?lb where {?a rdfs:label ?la. FILTER(lang(?la) = 'pt'). ?b␣
↪rdfs:label ?lb. FILTER(lang(?lb) = 'pt'). ?uri <http://dbpedia.org/property/
↪notableworks> ?a . ?uri <http://dbpedia.org/property/notableworks> ?b . ?uri a
↪<http://dbpedia.org/ontology/Writer> }"
```

3.4 Realizando uma consulta

Uma busca pode ser feita utilizando dois métodos presentes no QApedia, sendo eles o *QApedia.generator.perform_query()* e o *QApedia.generator.get_results_of_generator_query()*. O primeiro pode ser utilizado com qualquer consulta SPARQL, o segundo utiliza o primeiro método, mas antes ele ajusta a *generator_query* com a função explicada na seção *Ajustando a query geradora*. Como o resultado é grande, vamos apenas imprimir o tamanho da lista gerada e um exemplo.

```
>>> from QApedia import generator
>>> query = "select distinct ?a ?b where {\
...          ?uri <http://dbpedia.org/property/notableworks> ?a .\
...          ?uri <http://dbpedia.org/property/notableworks> ?b .\
...          ?uri a <http://dbpedia.org/ontology/Writer>}"
>>> results = generator.perform_query(query)
>>> len(results)
10000
>>> results[15]
{'a': Value(typed-literal:'Petty Crimes'), 'b': Value(typed-literal:'New and Selected␣
↪Poems')}
```

3.5 Geração de pares

A principal funcionalidade do pacote reside na geração de pares de questão-sparql a partir de um template previamente estabelecido, vamos definir um template inicial que retorna poucos resultados para critério de compreensão.

question	query	generator_query
Yoshihiro Togashi é autor de <A>?	ask where { <A> dbo:author dbr:Yoshihiro_Togashi }	select distinct ?a where { ?a dbo:author dbr:Yoshihiro_Togashi }

3.5.1 Realizando a consulta

Esse template é utilizado no formato de dicionário, onde cada chave corresponde ao nome da coluna, a chave *variables* corresponde as variáveis do tipo `?letra` localizadas entre *select* e o *where* da `generator_query`.

```
>>> template = {"question": "Yoshihiro Togashi é autor de <A>?",
...              "query": "ask where { <A> dbo:author dbr:Yoshihiro_Togashi }",
...              "generator_query": "select distinct ?a where { ?a dbo:author␣
↪dbr:Yoshihiro_Togashi }",
...              "variables": ["a"]}
>>> gquery = template["generator_query"]
>>> variables = template["variables"]
```

Dado esse template, inicialmente, iremos realizar o ajuste da *generator_query* adicionando o *label* do recurso armazenado na variável *a*. Para isso, realizamos a chamada da função de ajuste da `generator_query`:

```
>>> from QApedia.generator import adjust_generator_query
>>> gquery = adjust_generator_query(gquery, variables)
>>> print(gquery)
select distinct ?a ?la where {?a rdfs:label ?la. FILTER(lang(?la) = 'pt').  ?a␣
↪dbo:author dbr:Yoshihiro_Togashi }
```

Em seguida, pode ser utilizada a função de busca `perform_query` para realizar essa consulta. A consulta é realizada sobre a base da DBpedia, então não precisamos mudar valor padrão da função `endpoint="http://dbpedia.org/sparql"`.

```
>>> from QApedia.generator import perform_query
>>> results = perform_query(gquery)
>>> for instance in results:
...     print("%s: %s" %(instance["la"].value, instance["a"].value))
...
Level E: http://dbpedia.org/resource/Level_E
Yu Yu Hakusho: http://dbpedia.org/resource/Yu_Yu_Hakusho
Hunter × Hunter: http://dbpedia.org/resource/Hunter_×_Hunter
```

Outra forma de obter os resultados direto sobre a `generator_query` é utilizando o método `get_results_of_generator_query`.

```
>>> from QApedia.generator import get_results_of_generator_query
>>> generator_query = "select distinct ?a where { ?a dbo:author dbr:Yoshihiro_Togashi␣
↪}"
>>> variables = ["a"]
>>> results = get_results_of_generator_query(generator_query, variables)
>>> for instance in results:
...     print("%s: %s" %(instance["la"].value, instance["a"].value))
...
Level E: http://dbpedia.org/resource/Level_E
Yu Yu Hakusho: http://dbpedia.org/resource/Yu_Yu_Hakusho
Hunter × Hunter: http://dbpedia.org/resource/Hunter_×_Hunter
```

3.5.2 Construindo os pares

Para a geração dos pares de questão-sparql, utilizamos a função `extract_pairs`, ela recebe como parâmetro:

- **resultado** da busca no formato retornado pelo exemplo anterior
- **template** da busca
- **quantidade de pares** que você deseja gerar, se a busca retornar mais do que esse valor, ela é limitada por essa quantidade.

- **lista de prefixos**, caso deseje que os recursos retornados no formato `<http://dbpedia.org/...>` sejam substituídos pelo prefixo especificado.

Exemplo 1

```
>>> from QApedia.generator import extract_pairs
>>> pairs = extract_pairs(results, template, 2)
>>> for pair in pairs:
...     print(pair["question"])
...     print(pair["sparql"])
...     print("----")
...
Yoshihiro Togashi é autor de Level E?
ask where { <http://dbpedia.org/resource/Level_E> dbo:author dbr:Yoshihiro_Togashi }
----
Yoshihiro Togashi é autor de Yu Yu Hakusho?
ask where { <http://dbpedia.org/resource/Yu_Yu_Hakusho> dbo:author dbr:Yoshihiro_
↪Togashi }
----
```

Exemplo 2

```
>>> from QApedia.generator import extract_pairs
>>> from QApedia.utils import convert_prefixes_to_list
>>> prefixes = "PREFIX dbr:<http://dbpedia.org/resource/>\
...             PREFIX dbo:<http://dbpedia.org/ontology/>"
>>> list_of_prefixes = convert_prefixes_to_list(prefixes)
>>> list_of_prefixes
[('dbr:', 'http://dbpedia.org/resource/'), ('dbo:', 'http://dbpedia.org/ontology/')]
>>> pairs = extract_pairs(results, template, 2, list_of_prefixes)
>>> for pair in pairs:
...     print(pair["question"])
...     print(pair["sparql"])
...     print("----")
...
Yoshihiro Togashi é autor de Level E?
ask where { dbr:Level_E dbo:author dbr:Yoshihiro_Togashi }
----
Yoshihiro Togashi é autor de Yu Yu Hakusho?
ask where { dbr:Yu_Yu_Hakusho dbo:author dbr:Yoshihiro_Togashi }
----
```

Caso deseje substituir alguns símbolos da `sparql` por elementos textuais, você pode fazer isso através da função `encode`, para retornar a um formato válido novamente, basta utilizar o `decode`.

```
>>> from QApedia.utils import encode, decode
>>> for pair in pairs:
...     encoded = encode(pair["sparql"], list_of_prefixes)
...     decoded = decode(encoded, list_of_prefixes)
...     print(pair["question"])
...     print("====Encoded sparl====")
...     print(encoded)
...     print("====Decoded sparl====")
...     print(decoded)
...     print("----")
Yoshihiro Togashi é autor de Level E?
====Encoded sparl====
ask where  bracket_open  dbr_Level_E dbo_author dbr_Yoshihiro_Togashi  bracket_close
```

(continues on next page)

(continuação da página anterior)

```
====Decoded sparl====
ask where { dbr:Level_E dbo:author dbr:Yoshihiro_Togashi }
----
Yoshihiro Togashi é autor de Yu Yu Hakusho?
====Encoded sparl====
ask where  bracket_open  dbr_Yu_Yu_Hakusho dbo_author dbr_Yoshihiro_Togashi  bracket_
↪close
====Decoded sparl====
ask where { dbr:Yu_Yu_Hakusho dbo:author dbr:Yoshihiro_Togashi }
----
```

CAPÍTULO 4

QApedia package

4.1 Módulos

4.2 QApedia.generator module

O módulo `generator` permite ao usuário realizar buscas sobre o endpoint da dbpedia. Além disso, permite ao usuário realizar a construção de queries sparql dado um template previamente especificado.

Este arquivo pode ser importado como um módulo e contém as seguintes funções:

- adjust_generator_query - retorna a `generator_query` com os rótulos correspondente a cada variável.
- perform_query - realiza a execução da query no endpoint da dbpedia.
- get_results_of_generator_query - similar a função `perform_query`, entretanto, realiza os ajustes em cima da `generator_query` e salva o resultado da busca na memória.
- extract_pairs - realiza a construção dos pares de questão-sparql com base no resultado e template especificados.

`QApedia.generator.`**`adjust_generator_query`**(*generator_query*, *variables*, *lang='pt'*)

Dada uma `` `generator_query` `` é retornada uma versão contendo os labels que são utilizados para preencher as lacunas presentes na pergunta.

Parâmetros

- **`generator_query`** (*`str`*) – Query utilizada para geração dos pares de questão-sparql.
- **`variables`** (*`list`*) – Lista contendo as variáveis utilizadas nas lacunas da questão-sparql.
- **`lang`** (*`str, optional`*) – Idioma do campo `rdfs:label` adicionado na `generator_query`. O valor padrão é «pt».

Retorno Retorna a *generator_query* com os campos *labels* de cada variável.

Tipo de retorno str

Examples

No exemplo a seguir, temos a `generator_query` que será utilizada futuramente para retornar recursos que tenham o campo `dbo:abstract`. O resultado dela é usado para preencher as lacunas do seguinte par (`"o que é <A>?"`, `"select ?a where { <A> dbo:abstract ?a "`). Para preencher a lacuna da pergunta em linguagem natural, é adicionada na `generator_query` o campo `rdfs:label` correspondente as variáveis que se deseja obter informações.

```
>>> generator_query = "select distinct(?a) WHERE { ?a dbo:abstract []}"
>>> variables = ['a']
>>> result = adjust_generator_query(generator_query, variables)
>>> result
"select distinct(?a) ?la where { ?a rdfs:label ?la. FILTER(lang(?la) = 'pt').  ?a␣
↪dbo:abstract [] }"
```

`QApedia.generator.`**`perform_query`**(*query*, *prefixes=''*, *endpoint='http://dbpedia.org/sparql'*)

Dada uma query sparql retorna uma lista correspondendo ao resultado da pesquisa se a cláusula utilizada for SELECT, CONSTRUCT ou DESCRIBE. Caso seja ASK, o valor retornado é um *boolean*.

Parâmetros

- **`query`** (*`str`*) – Sparql utilizada para realizar uma busca no endpoint especificado.
- **`prefixes`** (*`str, optional`*) – Corresponde ao conjunto de prefixos utilizados na consulta SPARQL. Se não estiver usando prefixos, o uso desse parâmetro não é necessário, o valor padrão é «».
- **`endpoint`** (*`str, optional`*) – Indica endpoint utilizado, o valor default é `http://dbpedia.org/sparql`

Retorno

- *list of dict* – Corresponde a um lista contendo `bindinds` retornados pela busca Sparql. Se a cláusula utiliza SELECT ou CONSTRUCT.
- *bool* – Se a cláusula ASK for afirmativa retorna True, caso contrário False.

Examples

```
>>> from QApedia.generator import perform_query
>>> query = "SELECT * WHERE {"\
...         "?manga a dbo:Manga ."\
...         "?manga rdfs:label ?nome ."\
...         "?manga dbo:author dbr:Yoshihiro_Togashi ."\
...         "FILTER(lang(?nome) = 'pt').}"
>>> results = perform_query(query)
>>> for result in results:
...     print("%s: %s" %(result["nome"].value, result["manga"].value))
...
Level E: http://dbpedia.org/resource/Level_E
Yu Yu Hakusho: http://dbpedia.org/resource/Yu_Yu_Hakusho
Hunter × Hunter: http://dbpedia.org/resource/Hunter_×_Hunter
```

Raises `exc_type` – Caso haja um erro que não seja proveniente do problema de acesso ao endpoint, por exemplo, uma query em um formato inválido, uma exceção é gerada.

QApedia.generator.**get_results_of_generator_query**(*generator_query*, *variables*, *prefixes=''*, *endpoint='http://dbpedia.org/sparql'*, *lang='pt'*)

Dada uma `generator_query` é retornado um conjunto de resultados obtidos ao executar a query no endpoint especificado.

Parâmetros

- **generator_query** (*str*) – String representando a `generator_query`.
- **variables** (*list*) – Lista de caracteres correspondendo as variáveis.
- **prefixes** (*str, optional*) – Corresponde ao conjunto de prefixos utilizados na consulta SPARQL. Se não estiver usando prefixos, o uso desse parâmetro não é necessário, o valor padrão é «».
- **endpoint** (*str, optional*) – Indica endpoint utilizado., by default «http://dbpedia.org/sparql»
- **lang** (*str, optional*) – Idioma do campo `rdfs:label` adicionado na `generator_query`. O valor padrão é «pt».

Retorno

- *list of dict* – Corresponde a um lista contendo `bindinds` retornados pela busca Sparql. Se a cláusula utiliza SELECT ou CONSTRUCT.
- *bool* – Se a cláusula ASK for afirmativa retorna True, caso contrário False.

QApedia.generator.**extract_pairs**(*bindings*, *template*, *number_of_examples=500*, *list_of_prefixes=[]*)

Realiza a extração do conjunto de pares de questão-sparql correspondentes obtidos pelo método `QApedia.generator.get_bindings_of_generator_query()`.

Parâmetros

- **bindings** (*list*) – Resultado obtido após a execução da query correspondendo aos «bindings»
- **template** (*dict*) – Corresponde ao template utilizado para geração dos resultados.
- **number_of_examples** (*int, optional*) – Número de resultados a serem considerados para o template, o valor padrão é 500.
- **list_of_prefixes** (*list, optional*) – Corresponde a lista de prefixos obtida através do método *QApedia.utils.convert_prefixes_to_list()*, onde os prefixos devem ser os mesmos que foram utilizados na função que gerou os bindings. Se não estiver usando prefixos, o uso desse parâmetro não é necessário, o valor padrão é [].

Retorno Lista contendo os pares `sparql-question` do template.

Tipo de retorno list

Examples

```
>>> from QApedia.generator import extract_pairs
>>> from QApedia.generator import get_results_of_generator_query
>>> template = {"question": "Yoshihiro Togashi escreveu <A>?",
...             "query": "ask where {"\
...                      "dbr:Yoshihiro_Togashi ^ dbo:author <A>}",
```

(continues on next page)

(continuação da página anterior)

```
...             "generator_query": "select ?a where{"\
...                                "dbr:Yoshihiro_Togashi ^ dbo:author ?a}",
...             "variables": ["a"]}
>>> bindings = get_results_of_generator_query(
...                                         template["generator_query"],
...                                         template["variables"])
>>> pairs = extract_pairs(bindings, template)
>>> pairs[2]["question"]
'Yoshihiro Togashi escreveu Hunter × Hunter?'
>>> pairs[2]["sparql"]
'ask where {dbr:Yoshihiro_Togashi ^ dbo:author http://dbpedia.org/resource/Hunter_
↪×_Hunter}'
```

4.3 QApedia.io module

Este módulo trata das operações relacionadas a leitura e escrita do pacote `QApedia`.

Neste módulo, pode-se encontrar as seguintes funções:

- load_templates - realiza a leitura do arquivo contendo o conjunto de templates utilizados para a geração de perguntas-queries.

`QApedia.io.`**`load_templates`**(*filepath*, *delimiter=';'*)

A função load_templates, carrega o conjunto de templates a partir de um arquivo csv. O dado deve possuir um campo `generator_query` que servirá para realizar buscas que preencherão as lacunas presentes nos campos `question` e `query`.

Parâmetros

- **`filepath`** (*`str`*) – Caminho do arquivo csv que contém os templates.
- **`delimiter`** (*`str, optional`*) – Indicar qual separador utilizado no arquivo, by default ";"

Retorno Retorna um dataframe contendo o conjunto de templates.

Tipo de retorno pd.DataFrame

Examples

Exemplo contendo 14 templates sendo carregado através da função load_templates.

```
>>> from QApedia.io import load_templates
>>> filename = "sample.csv"
>>> templates = load_templates(filename)
>>> len(templates)
14
>>> templates.head()
                                              query  ... variables
0  <A> e <B> são os municípios vizinhos de que lu...  ...    [a, b]
1                <A> e <B> pertencem a qual espécie?  ...    [a, b]
2      <A> e <B> podem ser encontrados em qual país?  ...    [a, b]
3            <A> e <B> é produzido por qual empresa?  ...    [a, b]
4      <A> e <B> é o trabalho notável de qual autor?  ...    [a, b]

[5 rows x 4 columns]
```

4.4 QApedia.utils module

Este módulo contém o conjunto de operações utilizadas pelos módulos principais como por exemplo, o método *QApedia.io.load_templates()* que utiliza o método *QApedia.utils.extract_variables()* desse módulo para extrair o conjunto de variáveis presentes na query geradora (generator_query).

Neste módulo, pode-se encontrar as seguintes funções:

- extract_variables - realiza a extração das variáveis presentes no select da query geradora.
- convert_prefixes_to_list - dado o conjunto de prefixos, converte a string em uma lista de tuplas.
- encode - dada uma sparql realiza a codificação da query transformando alguns símbolos em texto.
- decode - dada uma sparql transformada por meio do encode realiza a transformação de inversa, de modo a substituir o texto por operações válidas.

QApedia.utils.**extract_variables**(*generator_query*)

Extrai as variáveis correspondente presentes no "generator_query".

Parâmetros **generator_query** (*str*) – A "generator_query" corresponde a query que será utilizada para obter as variáveis presente nas lacunas da pergunta(query) e do sparql.

Retorno Lista contendo as variáveis a serem respondidas.

Tipo de retorno lst

Examples

```
>>> generator_query = "select distinct ?a where {"\
...                   "?uri <http://dbpedia.org/ontology/author> ?a }"
>>> variables = extract_variables(generator_query)
>>> print(variables)
['a']
```

QApedia.utils.**convert_prefixes_to_list**(*prefixes*)

Converte uma string dos prefixos em uma lista de tuplas. Onde cada par contém um identificador e a uri correspondente.

Parâmetros **prefixes** (*str*) – string correspondendo aos prefixos utilizados na consulta SPARQL.

Retorno Lista de tuplas, onde cada tupla contém dois itens, o primeiro corresponde ao nome dado a URI que corresponde ao segundo item.

Tipo de retorno list

Examples

```
>>> from QApedia.utils import convert_prefixes_to_list
>>> prefixes = "PREFIX foaf: <http://xmlns.com/foaf/0.1/>\
...             PREFIX yago: <http://yago-knowledge.org/resource/>\
...             PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>\
...             PREFIX dbo:<http://dbpedia.org/ontology/>\
...             PREFIX dbp:<http://dbpedia.org/property/>"
>>> list_of_prefixes = convert_prefixes_to_list(prefixes)
>>> for prefix, uri in list_of_prefixes:
...     print(prefix, uri)
```

(continues on next page)

(continuação da página anterior)

```
...
foaf: http://xmlns.com/foaf/0.1/
yago: http://yago-knowledge.org/resource/
rdfs: http://www.w3.org/2000/01/rdf-schema#
dbo: http://dbpedia.org/ontology/
dbp: http://dbpedia.org/property/
```

QApedia.utils.**encode**(*sparql*, *prefixes*)

Dada uma query sparql, essa função transforma algum de seus caracteres em texto.

Parâmetros

- **sparql** (*str*) – sparql a ser transformada.
- **prefixes** (*list*) – lista de prefixos com uris utilizadas na sparql retornadas pela função *QApedia.utils.convert_prefixes_to_list()*.

Retorno sparql transformada.

Tipo de retorno str

Examples

```
>>> from QApedia.utils import encode
>>> from QApedia.utils import convert_prefixes_to_list
>>> prefixes = "PREFIX prop: <http://dbpedia.org/property/>\
...             PREFIX dbr: <http://dbpedia.org/resource/>"
>>> query = "ASK {\n\
...          <http://dbpedia.org/resource/Amazon_River> prop:length     ?amazon .
↪\n\
...          <http://dbpedia.org/resource/Nile> prop:length ?nile .\n\
...          FILTER(?amazon > ?nile) .\n\
...          }"
>>> list_of_prefixes = convert_prefixes_to_list(prefixes)
>>> query_encoded = encode(query, list_of_prefixes)
>>> print(query_encoded)
ASK  bracket_open
        dbr_Amazon_River prop_length     var_amazon  sep_dot
        dbr_Nile prop_length var_nile  sep_dot
        FILTER(var_amazon  greater_than  var_nile)  sep_dot
        bracket_close
```

QApedia.utils.**decode**(*sparql_encoded*, *prefixes*)

Dada uma sparql que foi codificada pela função *QApedia.utils.encode()*. O método `decode` substuir os termos codificados por símbolos válidos da consulta sparql.

Parâmetros

- **sparql_encoded** (*str*) – sparql transformada após passar por *QApedia.utils.encode()*.
- **prefixes** (*list*) – lista de prefixos com uris utilizadas na sparql retornadas pela função *QApedia.utils.convert_prefixes_to_list()*.

Retorno sparql com os símbolos válidos para uma consulta.

Tipo de retorno str

Examples

```
>>> from QApedia.utils import decode
>>> from QApedia.utils import convert_prefixes_to_list
>>> prefixes = "PREFIX prop: <http://dbpedia.org/property/>\
...             PREFIX dbr: <http://dbpedia.org/resource/>"
>>> list_of_prefixes = convert_prefixes_to_list(prefixes)
>>> query_encoded = "ASK  bracket_open \n\
...       dbr_Amazon_River prop_length var_amazon  sep_dot \n\
...       dbr_Nile prop_length var_nile  sep_dot \n\
...       FILTER(var_amazon  greater_than  var_nile)  sep_dot  \n\
...       bracket_close "
>>> print(decode(query_encoded, list_of_prefixes))
ASK {
    dbr:Amazon_River prop:length ?amazon .
    dbr:Nile prop:length ?nile .
    FILTER(?amazon > ?nile) .
    }
```

Índice de Módulos do Python

q

Índice