Zoo aggregation using semantic data

Safi Dewshi, 1559816

December 5, 2019

Contents

1 Introduction

This report will present and critique the ¡¡SOMETHING¿¿ of transferring a traditional relational data set into semantic data. Semantics is the study of meaning in language, and those codifying those meanings in a machine-readable format has several benefits for the field. For example, it allows programs to effectively "understand" what data means (Szeredi et al. 2014).

`http://cervo.io` The aim of this report is to present and evaluate the success of transferring a conventional SQL database into a SPARQL one (ibid.)

2 Concept and Aim

The intent of the project was to upgrade a SQL database into a SPARQL one that takes advantage of the interconnected nature of semantic data to allow more complex questions to be asked with simpler queries and to connect to external services to easily retrieve additional or updated data.

Figure 1 shows an illustration of the semantic web stack Starting from the bottom, these layers are:

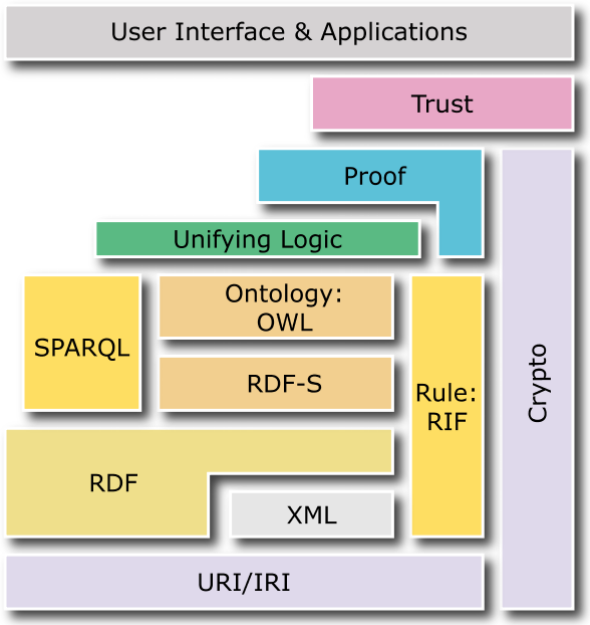

Figure 1: Semantic data "cake"(W3C n.d.)

1. URI/IRI, which are used for referencing and identifying resources
2. RDF/XML, which are used to store and represent information
3. SPARQL/OWL, which are used to infer ¡¡¿¿
4. Finally a layer for delivering that information to the user

It is planned for this application to completely replace the current SQL database that hosts the site

3 Design

3.1 Semantic Data Technologies

With semantic data

¡¡EXPLAIN TRIPLES¿¿

3.2 Ontology for cervo.io

Currently the website runs on a MySQL database, with several different tables containing the information needed to construct a taxonomic tree, populate it with species, and link those species to the zoos that keep them.

However this SQL solution had several problems that were easily remedied by switching to a SPARQL database:

It was inflexible and would require the schema to be rewritten to accommodate new types of data

Several other queries became significantly easier to write by taking advantage of the triples. Additionally the use of common keys such as postcode and species names makes the incorporation of external data sets significantly easier even within a single query

Additionally

¡¡EXPLAIN SPARQL¿¿

4 Implementation

The GUI was built using QT Designer, which produced a .ui file which was then referenced in the main python code

5 Evaluation and Use

6 Critical Reflection

7 Conclusion

References

Szeredi, Péter et al. (2014). *The Semantic Web Explained: The Technology and Mathematics behind Web 3.0*. Cambridge University Press. ISBN: 978-0-521-70036-8.

W3C (n.d.). *Semantic Web, and Other Technologies to Watch, slide 24*. URL: https://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(24).

Appendix

Appendix A

```
from PyQt5 import uic, QtWidgets
import sys
import requests
from PyQt5.QtWidgets import QAbstractItemView, QTableWidgetItem

Ui_MainWindow, QtBaseClass = uic.loadUiType("cervo.ui")
localurl = "http://localhost:3030/Cervo/query"
dbpedia = "http://dbpedia.org/sparql"
ordanancesurvey = "http://data.ordnancesurvey.co.uk/datasets/os-linked-data/apis/sparql"


class CervoUI(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)
        self.searchButton.clicked.connect(self.species_search)
        self.refineSearchButton.clicked.connect(self.refine_species_search)
        self.zooSearchButton.clicked.connect(self.zoo_search)
        self.refineZooSearch.clicked.connect(self.refine_zoo_search)
        self.currentSpecies = []
        self.currentZoos = []

    def species_search(self):
        searchtext = self.searchBar.text()
        print("Searching for: " + searchtext)
        url = localurl
        if self.visitableCheck.isChecked():
            sparql = """
            PREFIX cervo: <http://www.cervo.io/ontology#>

            SELECT ?name ?latinname
            WHERE
            {{{{
                ?animal cervo:hasLatinName ?latinname.
                ?animal cervo:hasName ?name.
                FILTER(EXISTS{{SELECT ?animal {{?animal ^cervo:keeps-animal|cervo:is-kept-at ?zoo.}}}})
                FILTER(REGEX(?name, "{}", "i"))
            }}
            UNION
            {{
                ?animal cervo:are-known-as|^cervo:can-refer-to ?collectiveterm.
                ?animal cervo:hasName ?name.
                ?animal cervo:hasLatinName ?latinname.
                FILTER(EXISTS{{SELECT ?animal {{?animal ^cervo:keeps-animal|cervo:is-kept-at ?zoo.}}}})
                FILTER(REGEX(?collectiveterm, "{}", "i"))
            }}}}

            """
        else:
            sparql = """
            PREFIX cervo: <http://www.cervo.io/ontology#>

            SELECT ?name ?latinname
            WHERE
            {{{{
                ?animal cervo:hasLatinName ?latinname.
                ?animal cervo:hasName ?name
```

```
            FILTER(REGEX(?name, "{}", "i"))
        }}
        UNION
        {{
            ?animal cervo:are-known-as|^cervo:can-refer-to ?collectiveterm.
            ?animal cervo:hasName ?name.
            ?animal cervo:hasLatinName ?latinname.
            FILTER(REGEX(?collectiveterm, "{}", "i"))
        }}}}
        """
    sparql = sparql.format(searchtext, searchtext)
    r = requests.get(url, params={'format': 'json', 'query': sparql})
    results = r.json()
    print(results)

    formatted_species = self.formatresults(results)
    self.currentSpecies = formatted_species

    self.format_view(formatted_species, self.speciesView)
    self.refineSearchButton.setEnabled(True)
    self.showZoos.setEnabled(True)
    self.showExtraInfo.setEnabled(True)
    self.speciesView.setCurrentCell(0, 0)

def refine_species_search(self):
    current_row = self.speciesView.currentRow()
    print(current_row)
    print(self.currentSpecies[current_row])

    if self.showZoos.isChecked():
        url = localurl
        sparql = """
        PREFIX cervo: <http://www.cervo.io/ontology#>

        SELECT ?zooname ?postcode ?zoo_website
        WHERE
        {{
            ?animal cervo:hasLatinName "{}".
            ?zoo cervo:keeps-animal|^cervo:is-kept-at ?animal.
            ?zoo cervo:hasName ?zooname.
            ?zoo cervo:hasPostcode ?postcode.
            ?zoo cervo:hasURL ?zoo_website
        }}
        """
    elif self.showExtraInfo.isChecked():
        url = dbpedia
        sparql = """
        PREFIX owl: <http://www.w3.org/2002/07/owl#>
        PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
        PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
        PREFIX foaf: <http://xmlns.com/foaf/0.1/>
        PREFIX dc: <http://purl.org/dc/elements/1.1/>
        PREFIX : <http://dbpedia.org/resource/>
        PREFIX dbpedia2: <http://dbpedia.org/property/>
        PREFIX dbpedia: <http://dbpedia.org/>
        PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
        SELECT ?name ?comment ?wikiarticle
```

```
            WHERE {{
             ?animal dbpedia2:taxon "{}"^^rdf:langString.
             ?animal dbpedia2:name ?name.
             ?animal rdfs:comment ?comment.
            FILTER (LANG(?comment)='en')
             ?animal ^foaf:primaryTopic ?wikiarticle
            }}
            """
        else:
            return

        sparql = sparql.format(self.currentSpecies[current_row]['latinname'])
        r = requests.get(url, params={'format': 'json', 'query': sparql})
        results = r.json()
        formatted_extra_data = self.formatresults(results)
        self.format_view(formatted_extra_data, self.extraInfoView)

    def zoo_search(self):
        searchtext = self.zooSearchBar.text()
        print("Searching for: " + searchtext)
        url = localurl
        sparql = """
        PREFIX cervo: <http://www.cervo.io/ontology#>

        SELECT ?zooName ?postcode ?zooWebsite
        WHERE
        {{
            ?zoo cervo:hasName ?zooName.
            ?zoo cervo:hasPostcode ?postcode.
            ?zoo cervo:hasURL ?zooWebsite
            FILTER(REGEX(?zooName, "{}", "i"))
        }}
        """
        sparql = sparql.format(searchtext)
        # todo: split the repeated section into its own function
        r = requests.get(url, params={'format': 'json', 'query': sparql})
        results = r.json()
        print(results)

        formatted_zoos = self.formatresults(results)
        self.currentZoos = formatted_zoos

        self.format_view(formatted_zoos, self.zooView)
        self.refineZooSearch.setEnabled(True)
        self.locationInfo.setEnabled(True)
        self.speciesKeptHere.setEnabled(True)
        self.zooView.setCurrentCell(0, 0)

    def refine_zoo_search(self):
        current_row = self.zooView.currentRow()
        print(current_row)
        print(self.currentZoos[current_row])

        if self.speciesKeptHere.isChecked():
            url = localurl
            sparql = """
            PREFIX cervo: <http://www.cervo.io/ontology#>
```

```
            SELECT ?speciesKept
            WHERE
            {{
                ?zoo cervo:hasPostcode "{}".
                ?zoo cervo:hasName ?zooName.
                ?zoo cervo:keeps-animal|^cervo:is-kept-at ?species.
                ?species cervo:hasName ?speciesKept
            }}
            """
        elif self.locationInfo.isChecked():
            url = ordanancesurvey
            sparql = """
            PREFIX owl: <http://www.w3.org/2002/07/owl#>
            PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
            PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
            PREFIX postcode: <http://data.ordnancesurvey.co.uk/ontology/postcode/>

            SELECT ?districtname ?countyname ?countryname
            WHERE {{
                ?postcodeUri rdfs:label "{}".
                OPTIONAL{{?postcodeUri postcode:district ?district.
                  ?district rdfs:label ?districtname}}.
                OPTIONAL{{?postcodeUri postcode:county ?county.
                  ?county rdfs:label ?countyname}}.
                OPTIONAL{{?postcodeUri postcode:country ?country.
                  ?country rdfs:label ?countryname}}
            }}
            """
        else:
            return

        sparql = sparql.format(self.currentZoos[current_row]['postcode'])
        r = requests.get(url, params={'format': 'json', 'query': sparql})
        results = r.json()
        formatted_extra_data = self.formatresults(results)
        self.format_view(formatted_extra_data, self.zooExtraInfo)

    def format_view(self, formatted_data, view):
        if len(formatted_data) == 0:
            formatted_data = [{'error': 'Query returned no results'}]
        row_count = (len(formatted_data))
        column_count = (len(formatted_data[0]))
        view.setColumnCount(column_count)
        view.setRowCount(row_count)
        view.setSelectionMode(QAbstractItemView.SingleSelection)
        view.setHorizontalHeaderLabels((list(formatted_data[0].keys())))
        for row in range(row_count):
            for column in range(column_count):
                item = (list(formatted_data[row].values())[column])
                view.setItem(row, column, QTableWidgetItem(item))

    def formatresults(self, data):
        keys = data['head']['vars']
        results = data['results']['bindings']
        return [{key: result[key]['value'] for key in keys} for result in results]


if __name__ == "__main__":
```

```
app = QtWidgets.QApplication(sys.argv)
window = CervoUI()
window.show()
sys.exit(app.exec_())
```