Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

# Project Report

## Submitted By Group A:

Tanmay Bagla – 19300702
Tanvi Bagla – 19300699
Mukesh Arambakam – 19301497
Shubham Dhupar – 19304374
Siddhartha Bhattacharyya - 19301936

## Description of Competency Questions:

The data sets used:
**Generalelection2016candidatedetails** - provides information of candidates and the parties to which they belonged to of the candidates who were contenders in the 2016 elections
**Generalelection2016constituencydetails** - provides details about the constituency like the quota, seats and vote information
**Generalelection2016countdetails** - provides details about the vote counting process.

Without all 3 datasets we would not have been able to answers questions combining count information and candidate's party, constituency and the candidate details, candidates and the deposit information from the constituency table, candidate's constituency and count details.

This was made possible because of combining the 3 datasets - forming rdfs and ontologies based on these so that we can ask such complex questions over the resulting linked data graph and defining relationships between classes of information.

## Description of Data Sets:

1. Candidate Details for General Election 2016
   This is information on the candidates for the General Election 2016, including name, gender, political party, votes received and their constituency.

   Column Details:

   - Constituency – Defines name of the constituency to which candidate belongs.
   - First Name, Surname, Gender – Defines details of the candidate.
   - Party – Defines party name to which candidate belongs.
   - Required to Save Deposit – Minimum number of votes required to get the deposit refund.
   - Votes – Number of votes received by candidate after the election.
   - Result – Election result of the candidate.
   - Constituency Number – Defines the constituent id of each candidate.

2. General Election 2016 Constituency Details
   This is a dataset of the Irish Constituencies in 2016, the number of seats, candidates, quota, total electorate and poll.

   Column Details:
   - Constituency Name – Defines Name of the Constituency.
   - Count Number – Defines the count iteration for final election results.
   - Number of Candidates – Defines number of candidates present a given constituency.
   - Number of Seats – Defines number of seats available in a given constituency.
   - Quota – Minimum number of votes to win an election from a given constituency.
   - Total Electorate – Total population of a given constituency.
   - Total Poll – Part of the total population who voted in an election.
   - Valid Poll – Defines total valid votes.
   - Spoiled – Defines invalid or rejected votes.
   - Constituency Number – Defines constituency ID.

3. General Election 2016 Count details-
   The number of votes received by each candidate in the General Election 2016.
   Column Details:

- Count Number – Defines each iteration count for election
- Occurred on Count - Defines the iteration count for final election results.
- Transfers – Defines the number of votes transferred from each candidate after election result is excluded.
- Constituency Number – Defines constituency ID.
- Candidate First Name, Candidate surname – Defines candidate details.

## Assumptions:

In “generalelection2016countdetails” dataset:

- We didn’t use Transfers, Non-Transferrable, required to reach quota columns.

In “generalelection2016constituencydetails” dataset:

- Date of Election was not used since the election happened on the same day.
- Seats in constituency & number of seats had the same data so only number of seats made it to uplifted dataset.
- Valid poll & Spoiled were also removed from uplifted dataset.

## References:

The following ontologies were reused:

- http://xmlns.com/foaf/spec
- www.data.gov.ie
- https://ukparliament.github.io/ontologies/

## Properties used:

The relationships identified in the classes were generally straightforward, and did not need much reasoning. However, we were able to identify some properties that were rendered effective by the reasoner in Protégé. The properties discovered are as follows:

**Inverse Property –** It is established between class Candidate & party since the candidate will always belong to a party & while the party has candidates.

**Symmetric Property –** This holds true for Candidate & Person since candidate is a person & person is a candidate so they are basically the same & hence symmetric.

**Transitive Property -** Candidate, Party & Constituency share a transitive relationship since Candidate belongs to a Party, Party belongs to Constituency. Hence, we can say that Candidate belongs to a Constituency.

## Data mapping process

**Data**: generalelection2016candidatedetails, generalelection2016constituencydetails, generalelection2016countdetails is taken from https://data.gov.ie/data.

**Data migration**: Data is uplifted by using JUMA Editor by defining the classes and its Subject, Predicate, Object.

**Classes Identified**- The main details that should be displayed directly to user and can be grouped together are classified into different classes. The 10 classes defined are as follows: Candidate, Party, Person, Result, Constituency, Electorate, Seats, Quota, Iterations, Votes

**Classes mapped to Subject:** Each class is mapped to a Subject with an IRI formed by Unique Ids. For example- ‘Candidate’ class is mapped to IRI https://www.electionsireland.org/2016/candidate/{Constituency Number}_{Candidate Id} where the instances in this class are filtered by unique ‘Constituency Number Candidate Id’.

**Classes mapped to Object:** Figuring out what data is related and can be grouped together under one class. Grouped the columns intending the similar details which are identified as Objects of that class.
For example –Person is made as one class containing the personal details of a candidate standing for election (First name, Last name and Gender).

**Subject to Object mapping:** Subject of each class is mapped to its objects by defining the Predicate.

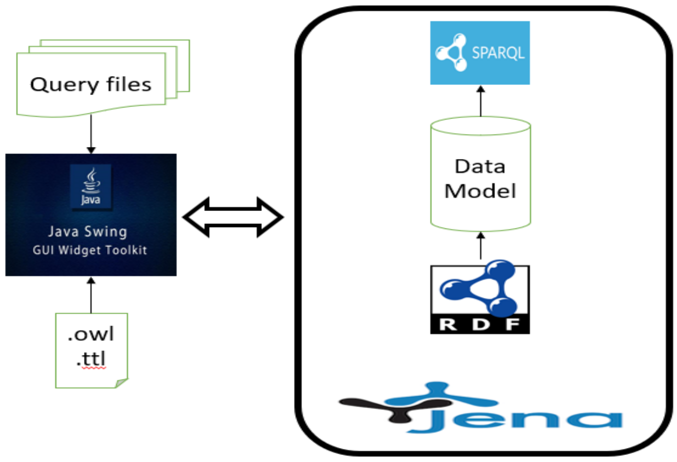
**Mapping of classes:** The classes are mapped to each other with different object properties by identifying the relationship between the classes.
For example- Class "Candidate" belongs to class "Constituency" i.e. for each Candidate there is a constituency from where he is standing. Details of 'constituency' class can be inherited by 'candidate' class.

| Classes | Subject IRI | Mapped to classes | Objects in each class |
|---|---|---|---|
| Candidate | https://www.electionsireland.org/2016/candidate/{Constituency Number}_{Candidate Id} | Party, Iterations, Person, Result, Constituency | Candidate ID |
| Party | https://www.electionsireland.org/2016/candidate/party/{PARTY} | Candidate | Party Abbreviation, Party |
| Person | https://xmlns.com/foaf/0.1/{First Name}_{SURNAME} | Candidate | Surname, Firstname, Gender |
| Result | https://www.electionsireland.org/2016/candidate/{Constituency Number}_{Candidate Id}_{RESULT} | Candidate | Result |
| Constituency | https://www.electionsireland.org/2016/constituency/{Constituency Number} | Electorate, Seats, Quota, Candidate | Constituency Name, Number of Candidates, Count Number |
| Electorate | https://www.electionsireland.org/2016/constituency/electorate/{Total Electorate} | Constituency | Total Poll, Valid Poll, Spoiled |
| Seats | https://www.electionsireland.org/2016/constituency/seats/{Seats in Constituency} | Constituency | Seats Filled |
| Quota | https://www.electionsireland.org/2016/constituency/quota/{QUOTA} | Constituency | Required to save deposit |
| Iterations | https://www.electionsireland.org/2016/count/{Constituency Number}_{Candidate Id}_{Count Number} | Votes, Candidate | Occurred on Count, Result, Count Number |
| Votes | https://www.electionsireland.org/2016/count/votes/{Constituency Number}_{Candidate Id}_{Count Number} | Candidate | Transfers, Votes, Total Votes |

## Application Query Interface

The query interface has been designed using Java Swing framework, which is a GUI widget toolkit for Java. This Java API provides a sophisticated set of GUI components that can be used to present simple out of the box UI features such as buttons, text areas, scroll-panes etc. It has been integrated with Apache Jena to support loading turtle and owl files and also provides features to query over the data sets imported via SPARQL. As Apache Jena provides for a seamless framework for querying, loading and creating ontology models for Java applications, Swing was the obvious choice for the front end.
A predefined set of queries have been saved in a local directory folder which are loaded onto the UI as the application is launched. Java Swing provides drop down menus from which a query can be selected to be executed. The queries have been defined as per the ontology designed. The knowledge file which can be either .owl or .ttl is imported onto the UI. Apache Jena converts the knowledge from the file into an ontology and thus the data model is created. With

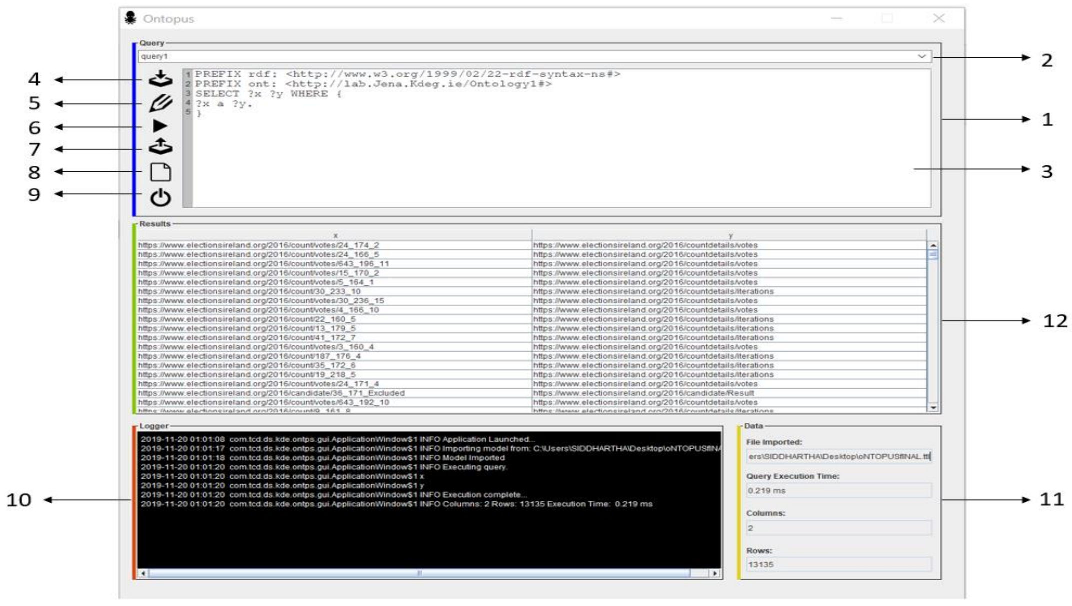
the help of the SPARQL engine integrated in Apache Jena, a query can be selected from the UI and executed on the data model created. The basic architecture of the application is described below:



Query execution over Apache Jena proved be more efficient compared to other resources available like those in python namely ‘SPARQL Wrapper’. An average execution speed of 0.3 ms was achieved over our combined data set of about 5000 rows. The UI interface has not been restricted to only work on the model designed for our data set. It has been generalized in a fashion that allows any model to be imported and attached to a session. Similarly, the back end SPARQL engine has been customized and generalized to return the exact columns that are being queried.

## UI Features

The UI has been built keeping in mind the simplicity and the basic requirement of the project. Only the most relevant and important features have been implemented keeping our application understandable. The basic features and buttons have been described below.



```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ont: <http://lab.Jena.Kdeg.ie/Ontology1#>
SELECT ?x ?y WHERE {
?x a ?y.
}
```

| x | y |
|---|---|
| https://www.electionsireland.org/2016/count/votes/24_174_2 | https://www.electionsireland.org/2016/countdetails/votes |
| https://www.electionsireland.org/2016/count/votes/24_166_5 | https://www.electionsireland.org/2016/countdetails/votes |
| https://www.electionsireland.org/2016/count/votes/643_196_11 | https://www.electionsireland.org/2016/countdetails/votes |
| https://www.electionsireland.org/2016/count/votes/15_170_2 | https://www.electionsireland.org/2016/countdetails/votes |
| https://www.electionsireland.org/2016/count/votes/5_164_1 | https://www.electionsireland.org/2016/countdetails/votes |
| https://www.electionsireland.org/2016/count/30_233_10 | https://www.electionsireland.org/2016/countdetails/iterations |
| https://www.electionsireland.org/2016/count/votes/30_236_15 | https://www.electionsireland.org/2016/countdetails/votes |
| https://www.electionsireland.org/2016/count/votes/4_166_10 | https://www.electionsireland.org/2016/countdetails/votes |
| https://www.electionsireland.org/2016/count/22_160_5 | https://www.electionsireland.org/2016/countdetails/iterations |
| https://www.electionsireland.org/2016/count/13_179_5 | https://www.electionsireland.org/2016/countdetails/iterations |
| https://www.electionsireland.org/2016/count/41_172_7 | https://www.electionsireland.org/2016/countdetails/iterations |
| https://www.electionsireland.org/2016/count/votes/3_160_4 | https://www.electionsireland.org/2016/countdetails/votes |
| https://www.electionsireland.org/2016/count/187_176_4 | https://www.electionsireland.org/2016/countdetails/iterations |
| https://www.electionsireland.org/2016/count/35_172_6 | https://www.electionsireland.org/2016/countdetails/iterations |
| https://www.electionsireland.org/2016/count/19_218_5 | https://www.electionsireland.org/2016/countdetails/iterations |
| https://www.electionsireland.org/2016/count/votes/24_171_4 | https://www.electionsireland.org/2016/countdetails/votes |
| https://www.electionsireland.org/2016/candidate/36_171_Excluded | https://www.electionsireland.org/2016/candidate/Result |
| https://www.electionsireland.org/2016/count/votes/643_192_10 | https://www.electionsireland.org/2016/countdetails/votes |

```
2019-11-20 01:01:08  com.tcd.ds.kde.ontps.gui.ApplicationWindow$1 INFO Application Launched...
2019-11-20 01:01:17  com.tcd.ds.kde.ontps.gui.ApplicationWindow$1 INFO Importing model from: C:\Users\SIDDHARTHA\Desktop\oNTOPUSfINA
2019-11-20 01:01:18  com.tcd.ds.kde.ontps.gui.ApplicationWindow$1 INFO Model Imported
2019-11-20 01:01:20  com.tcd.ds.kde.ontps.gui.ApplicationWindow$1 INFO Executing query.
2019-11-20 01:01:20  com.tcd.ds.kde.ontps.gui.ApplicationWindow$1 x
2019-11-20 01:01:20  com.tcd.ds.kde.ontps.gui.ApplicationWindow$1 y
2019-11-20 01:01:20  com.tcd.ds.kde.ontps.gui.ApplicationWindow$1 INFO Execution complete...
2019-11-20 01:01:20  com.tcd.ds.kde.ontps.gui.ApplicationWindow$1 INFO Columns: 2 Rows: 13135 Execution Time:  0.219 ms
```

File Imported: ers\SIDDHARTHA\Desktop\oNTOPUSfINAL.ttl
Query Execution Time: 0.219 ms
Columns: 2
Rows: 13135

1. The query interface sections which includes the control panel and the query editor.
2. Drop down menu listing all the queries.
3. Query editor pane – enables undo and redo features.
4. Import button – Opens local machine’s directory structure so that the ontology file can be selected and imported.
5. Save Query button – In-case a change needs to be saved to a query file, this button will save the current text on the editor pane to the file selected from the drop down menu.
6. Execute button – Executes the query in the editor pane on the ontology model that has been loaded.

7. Export button – Exports the result-set of a query.
8. Clear button – clears and refreshes the screen. Leaves the ontology attached to the session.
9. Power button – Exits and closes the application.
10. Logger window – shows the logs of the application, useful in case of any exceptions.
11. Data window – Displays the ontology attached and its corresponding file location, number of rows returned but the query, number of columns returned by the query, along with the execution time.
12. Result window – displays the query result set.

## Description of Queries

The 10 queries developed have been modeled keeping 3 essential points in mind:

- All queries return values that are derived from a combination of all three data sets.
- Include most SPARQL features in query
- Model queries that are otherwise unanswerable from single data set.

The queries have been described below:

1. *From which constituency max. number of females got elected?* - The query finds the count of all the female candidates who got elected per constituency and displays the maximum of the count.
   *SPARQL features used*: filter, count, distinct, group by, order by, limit

2. *Name the parties who won from Dublin (all constituencies in Dublin)* – Party names are returned based filtering over the result being equaled to ‘Elected’ and a regular expression for finding all constituencies whose name contains ‘Dublin’.
   *SPARQL features used*: filter, regex, group by.

3. *Count the no. of males who got elected from Cork* - Count of all the distinct candidates are returned after applying a filter of the person being a male, the result being 'Elected' and a regex filter that he is from Cork.
   *SPARQL features used*: filter, regex, group by, count

4. *List the candidates in a constituency having min. no. of candidates*- The query involves subqueries where the inner query returns the constituency which has the minimum number of candidates and the outer query uses the result from the inner query and finds the candidates who belong to that constituency and lists there first name and last name.
   *SPARQL features used*: subquery, group by, order by, limit, count, distinct

5. *Find the candidates who lost their deposits in Louth*- Applies filters on constituency name being Louth, and the result being 'Excluded'.
   *SPARQL features used*: filter, regex.

6. *Find the no of candidates who got excluded after the 5th round of count*- This query finds the candidates who got excluded from the counting process by filtering over the 'Has Occurred On' column and the result.
   *SPARQL features used*: filter, count, distinct

7. *Find the party/parties who have the maximum number of elected candidates*- Applies filter over the candidates result and then finds their party and list the party who have the maximum number of elected candidates.
   *SPARQL features used*: filter, count, group by, order by, limit.

8. *Find the min no of votes a candidate has won the election with*- Queries over the candidates who got elected and finds the minimum of the total votes to find the required candidates.
   *SPARQL features used:* filter, min

9. *Find the parties who got elected from Louth but got excluded from 'Dublin West'*- The query applies Union over two queries the first which return the parties who lost in Dublin West and the second query returns the

parties who won in Louth.
*SPARQL features side*: filter, Union, regex

10. *Find the party/parties which has won most of the seats in constituency whose total poll is greater than 70k -* This query applies filter over the total poolers being over 70000 and the result of the candidates being elected and finds the party to which the candidates belongs too and list the party names after grouping them.
    *SPARQL features used*: filter, group by

## Challenges Faced while ontology modelling and mapping:

It was a challenge to find unique column in the dataset. Finally used combination of Candidate ID and Constituency number.

- It was difficult to find online resources on Juma.
- Initially it was difficult to define relationships between classes and implement different properties such as Transitive and Symmetric.
- Finding out the unique Ids for each class that work as primary ids for class to class mapping.
- Have to update the data mapping repeatedly while adding new classes or objects in the class.
- Ensuring that the data mapping should not make the extraction of data through queries, challenging.

## Conclusion:

The application, data set and ontology have been developed keeping in mind that the aim is to answer questions that are meaningful and relatable to the data set selected. The elections data set was chosen for a reason as it is always helpful to have legacy data and information on political proceedings as it might be useful for future candidates to design campaigns.

- Three datasets were chosen which had well established relationships but were not explicitly defined.
- The SPARQL queries created using these combined datasets helped provide solutions to real world problems. Moreover, the ontology helped the team to understand the elections system in Ireland.
- Initially it was difficult understanding the tools such as Juma and Protege due to limited online resources but after group brainstorming session it was easy to proceed.
- Given the opportunity more UI features could be included to develop a well-rounded application that may be able to imitate industry standards.

## Self-Reflection:

### Mukesh

***Contributions**:* Worked on developing competency questions and SPARQL queries after analyzing data sets. Also helped out with data mapping and understanding the meaning of columns in the data set.
***Strengths/Weaknesses**:* Learnt the importance of RDF creations and Ontology modelling, and how inconsistencies in creations of these classes and data properties result in absurd results in questioning the ontology. I worked on questioning the datasets and creating the queries using SPARQL, and I experienced this first hand when we missed an link in the rdf creation. We had to rebuild the rdf. But then upon doing so we understood the importance of linked data creation and how things knowledge gets built using multiple data sets, and reasoning works in an Ontology. SPARQL queries were a bit difficult to write as we had to make sure we used the proper predicate which provides meaning to the question we asked.

### Tanmay

***Contributions**:* Worked on finding three open datasets related to Election Ireland which could be explained reasonably well through the defined ontology. Worked with Shubham to define classes, object properties and data properties while creating ontology in Protege.

***Strengths/Weaknesses****:* Learnt to create ontology in protégé within less time frame through online resources. Actively participated in group discussions while writing SPARQL queries to remain on the set target. Not able to provide in depth contribution on creating UI and uplifting dataset on Juma due to time constraints.

## Shubham –

***Contributions****:* Worked with Tanmay to define the classes and the properties. Also worked on creating the owl ontology.
***Strengths/Weaknesses****:* Everyone in the group synced well from the beginning. The work was evenly divided among all us & helped each other when any issue was encountered. I didn’t have any hands-on JUMA & Protégé so I took a lot of time to understand the tool.

## Tanvi

***Contributions****:* Worked on finding three open datasets related to Election Ireland which could be explained reasonably well through the defined ontology. Worked with Shubham to define classes, object properties and data properties while creating ontology in Protege.
***Strengths/Weaknesses****:* Learnt to create ontology in protégé within less time frame through online resources. Actively participated in group discussions while writing SPARQL queries to remain on the set target. Not able to provide in depth contribution on creating UI and uplifting dataset on Juma due to time constraints.

## Siddhartha

***Contributions****:* Worked on developing the User Interface and developing a solution to generalize SPARQL query back end system to handle any form of query on a specific knowledge file. Also worked on developing mapping and relationships among different data sets.
***Strengths/Weaknesses:*** Given the opportunity more UI features could be included to develop a well-rounded application that may be able to imitate industry standards. The UI is generalized and scales well over a variety of queries and also achieves efficient execution throughput. Also, the mapping task helped me get an understanding of the importance of linked data and how mapping different data helps us to answer complex questions.