
ECS7028U/ECS7028P: Company Ontology Report

Student Name: Sahib Bhatti

Student ID: 190319889

Contents

1. Introduction.....	2
2. Core Task	2
3. Extra Task.....	13
4. Advanced Task	15
5. Sparql Queries	18
6. Advanced Queries	20
.....	20
7. Issues and Resolutions.....	20
8. Conclusion.....	21
9. List of Files	22

1. Introduction

This report is about the design, implementation, and explanation of my Company ontology (IRI: <http://www.semanticweb.org/ontologies/user/Company>). This ontology models the structure of the business entities, their relationships, and associated data properties. This report covers the core and advanced tasks describing the creation of basic taxonomy property hierarchy and reasoning with SWRL rules where we used The Hermit 1.3.8 and Pellet reasoner in Protégé to check for inconsistencies.

2. Core Task

2.1 Correctly Designed Ontology with Basic Taxonomy and Property Hierarchy

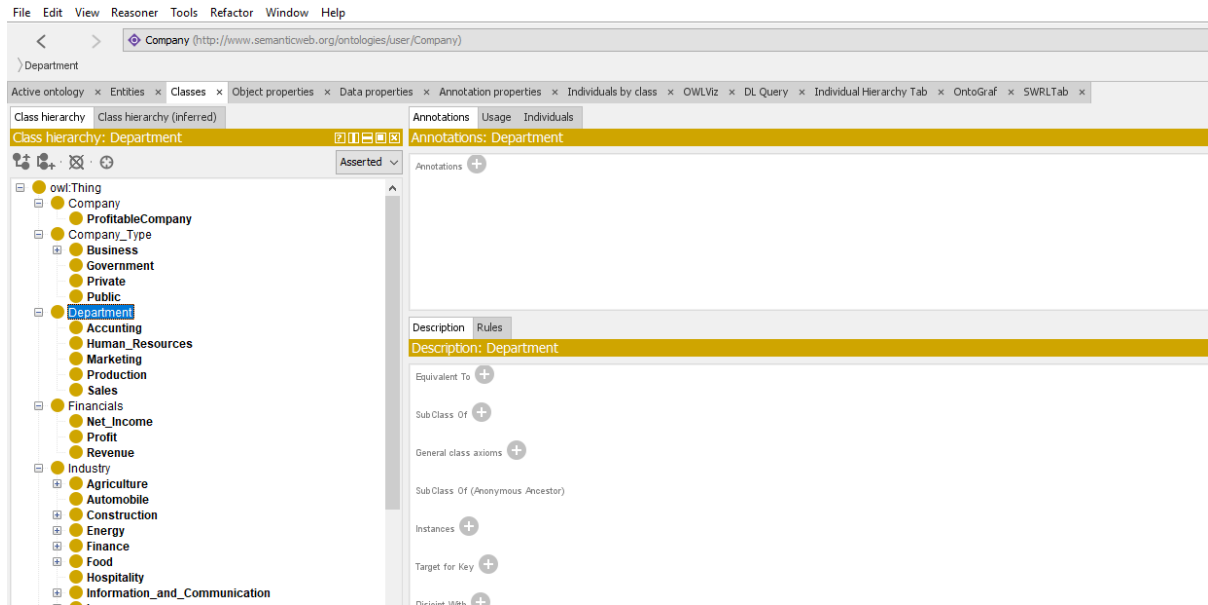
The ontology's taxonomy consists of various classes and subclasses that represent different aspects of a company. The following is an overview of the core structure of the company ontology:

- **Classes and Subclasses:** The primary classes in the ontology are **Company**, **Persons**, **Product**, **Service**, **Location**, **Industry**, **Industry_type**, **Company_type**, **Name**, **Financials**, **Departments**. Subclasses are used to categorize entities further, like **Business**, **Government**, **Public**, **Private** for companies, subclasses like **Accounting**, **Human_Resource**, **Marketing**, **Production**, **Sales** for the **Department Class**, Subclasses like **Net_income**, **Profit**, **Revenue** for **Financials Class**, Multiple subclasses for **Industry** class which include **Agriculture**, **Automobile**, **Construction** with subclasses **Builder**, **Contractor**, **Plumbing**, **Energy**, **Finance**, **Food**, **Hospitality**, **Mining**, **Real_Estate** etc, **City** and **Country** subclass for **Location** class. **CEO**, **Intern**, **Manager** and **President** subclass for the **Person Class**. **Main_Products** and **Associated_Products** subclass for the **Products** class.
- **Asserted and Inferred Class**

Of all these classes I defined above, **Company**, **Persons**, **Products**, **Location** **Departments**, **Financials** are asserted (Primitive classes). The rest are inferred such as **HighRevenue** or **Low Revenue Company**, **Large Company** or **Small Company**, **Associated Products** or **Main_Products** are also inferred class. **CEO** class in the **Person** is also inferred class based on the role of the company staff.

Class	Asserted	Inferred
Company	Government, Private, Public, Business	Is a large or small company based on the number of employees. Is a profitable company or not based on profit they make.
Persons	CEO, Intern, Manager, Staff	Is a CEO or not based on role, is a Junior Staff or Senior Staff based on age.
Products	Main_Products, Associated_Products	Company with the specific products

Service	Banking, Communication, Construction	N/A
Location	City, Country	Is the company located in the specific city/Country or not
Industry	Agriculture, Real_Estate, Manufacturing, Insurance,	List of Industry Types
Departments	List of Departments	N/A



- **Property Hierarchy:** Object and data properties are structured logically. Object properties connect different classes according to the relationships, and data properties represent attributes or data points.

2.2 Correct Domain and Range Restrictions

Each object property in the ontology has defined domain and range restrictions. This section covers of these restrictions and their rationale:

- **Object Properties:**

Object Property	Domain	Range	Reason
Address	Company	Location	Each Company has an address where it is located
Associated_with	Products	Company	Each product is associated to the company
CEO	Company	CEO	Each company have a CEO and it only belongs to the list of the CEOs

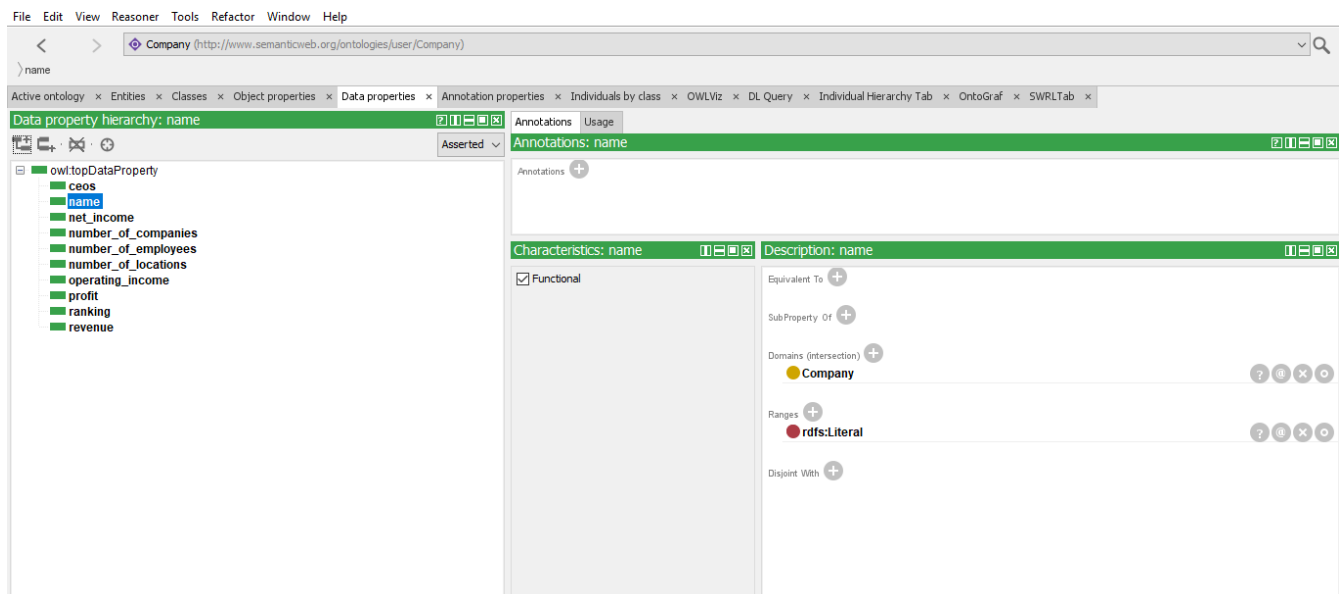
City_located	Company	City	Each company is in some city
Country_located	Company	Country	Each company is in some country
Headquarter	Company	Country	Each company has headquarter and it located in country
Industry_Type	Company	Industry	Each company belongs to the single or multiples industry
Manufactured_By	Products	Company	Products are manufactured by different companies.
Service_provided	Company	Service	Each company provides multiple services

The screenshot shows the Protégé ontology editor interface. The top menu bar includes File, Edit, View, Reasoner, Tools, Refactor, Window, and Help. The main window displays the 'Company' ontology (http://www.semanticweb.org/ontologies/user/Company/). The left sidebar shows the 'Object property hierarchy: ceo_of' with a list of properties including address, associated_with, ceo, ceo_of, city_hosts, city_located, country_hosts, country_located, hasAge, headquarter, headquarter_of, industry_type, industry_type_companies, industry_type_of, manufactured_by, manufactures, service_provided, and service_provided_by. The right pane shows the 'Annotations: ceo_of' tab, which is currently empty. Below it, the 'Characteristics' tab is active, showing the 'Description: ceo_of' with various property characteristics (Functional, Inverse functional, Transitive, Symmetric, Asymmetric, Reflexive, Irreflexive) and a list of domains (CEO, Company) under the 'Domains (intersection)' section.

- **Data Properties:**

Data Property	Domain	Range	Reason
CEO	CEO	Literal	CEO belongs to the ceos class and range is literal
Name	Company	Literal	Each company contains some name
Net_income	Company	Double	Company have net_income

			generated each month or year
Number_of_companies	Company	Integer	Total numbers of companies in the all countries
Number_of_Employees	Company	Integer	Each company have any number of employees
Number_of_locations	Company	Integer	Each company located in each country
Operating_income	Company	Double	Each company have operating income to operates all the operations
Profit	Company	Double	Profit generated by the company after selling its products
Ranking	Company	Integer	Ranking of the company in the world



Asserted and Inferred Properties:

We asserted the following Datatype properties:

- ceos (?c, ?a): Company ?c has CEO ?a. CEO is an Literal.
- name (?c, ?n): Company ?c has name ?n. Name is a string.
- net_income(?c, ?nic): Company ?c generates ?nic net Income. ?nic is an double.

- number_of_companies(?c, ?len): Company ?c has ?len companies. ?len is an integer.
- number_of_employees(?c, ?emp): Company ?c has ?emp employees. ?emp is an integer.
- number_of_locations(?c, ?loc): Company ?c has ?loc locations. ?loc is an integer.
- profit(?c, ?pt): Company ?c makes ?pt profit. ?pt is an double.
- ranking(?c, ?ra): Company ?c has ?ra rankings. ?ra is an integer.
- revenue(?c, ?re): Company ?c makes ?re revenue. ?re is an double.

We asserted the following Object properties:

- address(?c, ?add): Company ?c has address ?add.
- ceo_of(?c, ?co): Company ?c have ceo ?co.
- city_located(?c, ?cl): Company ?c located in city ?cl.
- company_Number(?c, ?number): Company ?c have unique ?number.
- company_type(?c, ?type): Company ?c have some type ?type.
- country_hosts(?c, ?ch): Company ?s located in country ?ch.
- has_Age(?p, ?age): Person ?p has age ?age.
- headerquarter(?c, ?hq): Company ?c have headquarter ?hq.
- industry_type(?c, ?it): Company ?c has industry_type ?it.
- manufactures(?c, ?mp): Company ?c manufactures products ?mp.
- service_provided(?c, ?sp): Company ?c provides services ?sp.

We inferred the following Object Properties:

- ceo(?ce, ?c): Inverse of ceo_of. CEO ?ce holds on ?c.
- headerquarter_of(?hq, ?c): Inverse of headerquarter(?hq, ?c). Headerquater ?hq belongs to company ?c.
- manufactured_by(?mp, ?c): Inverse of manufactures(?c, ?mp). products manufactured ?mp by company ?c.
- service_provided_by(?sp, ?c): Inverse of service_provided(?c, ?sp). Service provided ?sp by company ?c.

2.3 Correct and Effective Use of Object Properties Including Constraints and Characteristics

Object properties in the ontology connect different classes and define relationships. This section describes how these properties are used and any constraints or characteristics applied:

- **Constraints and Characteristics:** Properties like **address** are functional, indicating an address cannot be same. The **associatedWith** property is symmetric, showing circular relationships like Each company is associated to some industry or manufacturing product.

The **ceo_of property** is inversely functional showing they can work inversely or belongs to same classes.

- **Effective Use of Object Properties:** Properties are used to establish meaningful relationships, such as a company being located in a city and having a headquarters in a country.

2.4 Correct and Effective Use of Data Properties

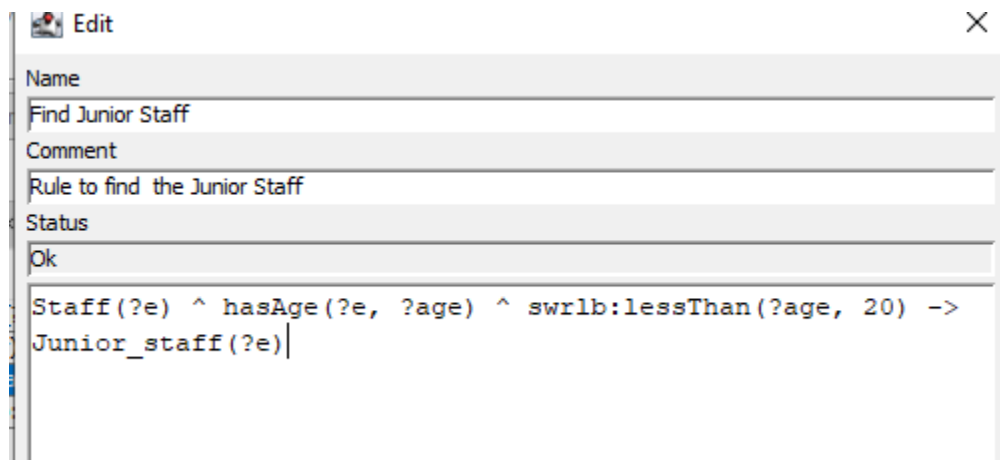
Data properties in ontology represent various attributes associated with classes. Here's an explanation of how these properties is used and why they are effective:

- **Examples of Data Properties:** The **profit** property represents a company's annual profit, while the **number_of_employees** property indicates the number of employees. For a complete view of data properties, I insert the table upper.
- **Use of Data Properties:** These data properties are used to categorize companies based on revenue and profit generated, categories the companies or employees based on strength of the employee, calculate ratios, and create SWRL rules based on specific data values.

2.5 Logical and Correct Use of Description Logic to Define Concepts

The ontology uses Description Logic (DL) to define concepts and relationships. This section provides examples of how DL is used to create logical structures and ensure correct relationships:

- **Defining Concepts:** Subclasses are defined logically, such as **SeniorEmployee** or **JuniorEmployee** for employees aged 20 and older. These relationships are derived through DL constructs like **subClassOf**.



Dialog box titled "Edit" showing rule configuration:

- Name: Find Junior Staff
- Comment: Rule to find the Junior Staff
- Status: Ok
- Rule text: `Staff(?e) ^ hasAge(?e, ?age) ^ swrlb:lessThan(?age, 20) -> Junior_staff(?e)`

- Also, subclass is defined locally such as **ProfitableCompany** if company generate the profit more than some fixed amount. For our case it is 1000000. These relationships are derived through DL constructs like **subclassOf**.

Name	ProfitableCompany
Comment	Rule to find ProfitableCompany if profit greater than some limit
Status	Ok
<pre>Company(?comp) ^ profit(?comp, ?p) ^ swrlb:greaterThan(?p, 1000000) -> ProfitableCompany(?comp)</pre>	

DL and SWRL rules for inferred classes and properties:

Rules for classes can be found below. Some of rules are implemented directly as DL rules, others as SWRL rules, but for consistency I write all the rules using DL and SWRL Syntax.

Class	Rule
Junior Staff	<code>Staff(?e) ^ hasAge(?e, ?age) ^ swrlb:lessThan(?age, 20) -> Junior_staff(?e)</code>
Senior Staff	<code>Staff(?e) ^ hasAge(?e, ?age) ^ swrlb:lessThan(?age, 20) -> Senior_staff(?e)</code>
Large Company	<code>Company(?c) ^ number_of_employees(?c, ?count) ^ swrlb:greaterThan(?count, 500) -> LargeCompany(?c)</code>
Small Company	<code>Company(?c) ^ number_of_employees(?c, ?count) ^ swrlb:lessThanOrEqual(?count, 500) -> SmallCompany(?c)</code>
Profitable Company	<code>Company(?comp) ^ profit(?comp, ?p) ^ swrlb:greaterThan(?p, 1000000) -> ProfitableCompany(?comp)</code>

Executive Lean	Company Company(?c) ^ ceos(?c, ?e) ^ number_of_employees(?c, ?count) ^ swrlb:divide(?ratio, ?count, ?e) ^ swrlb:lessThanOrEqual(?ratio, 10) -> ExecutiveLeanCompany(?c)
Agriculture	Industry and (industry_type some Agriculture)
CEO	Company(?comp) ^ ceo(?comp, ?ce) -> CEO(?ce)
Products	Products and (associated_with some Main_Products)
Location	Location and (city_located some City)
Retail	Industry and (industry_type some Retail)

Object and Data Property

Rule

City_located	Company(?c) ^ city_located(?c, ?city) ^ city_located(?city, ?country) -> city_located(?c, ?country)
Headquarter	headquarter \equiv country_located
Address	address \equiv city_located
associated_with	associated_with \equiv manufactured-by-
CEO	has-CEO \equiv CEO
industry_type	industry_type \equiv belongs-to- industry
Revenue and Profit	profit \subseteq revenue
Number of Employees	number_of_employees \geq 1
Operating Income	operating_income \subseteq revenue

For the object property, I use both SWRL and DL rules format because some rules are not implemented using DL format because the pallet reasoner does not support them.

2.6 Ontology Population Python Script and SPARQL Queries

This section describes the process used to populate the ontology with data and the SPARQL queries employed to retrieve specific information:

- **Python Script:** A script was used to fetch data from various sources such as DBPedia, then convert it to RDF, and import it into the ontology according to our design and structure. When we fetch the data from DBPedia, it also extracts irrelevant data from the external resources. So, for extracting the relevant data from RDF we use the sparql queries to fetch specific information.
- **SPARQL Queries:** Multiple Queries were used to extract relevant information from the RDF data and insert it into the ontology, allowing for easy population and data retrieval.

Example

```
PREFIX org: <http://www.semanticweb.org/ontologies/user/Company.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX dbpprop: <http://dbpedia.org/property/>

CONSTRUCT {
    ?company rdf:type org:Company .
    ?company org:name ?name .
    ?company org:ceo ?ceo .
    ?ceo rdf:type org:Ceos .
    ?company org:locationCity ?locationcity .
    ?locationcity rdf:type org:City .
    ?company org:headquarter ?headquarter .
    ?headquarter rdf:type org:Country .
    ?company org:industry ?industry .
    ?industry rdf:type org:Industry_Type .
    ?company org:product ?product .
    ?product rdf:type org:Main_Products .
    ?company org:service ?service .
    ?company org:netIncome ?profit .
    ?company org:companyType ?companyT .
    ?company org:numberOfEmployees ?number_of_employees .
    ?company org:operatingIncome ?operating_income .
    ?company org:operatingIncome ?operating_income .
    ?company org:ranking ?ranking .
    ?company org:revenue ?revenue .
}

WHERE{
    ?company rdf:type dbpedia-owl:Company .
    ?company foaf:name ?name .
```

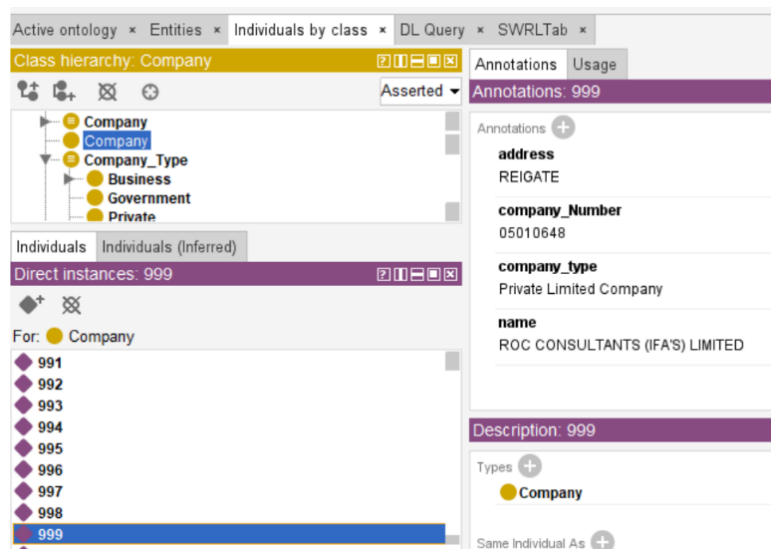
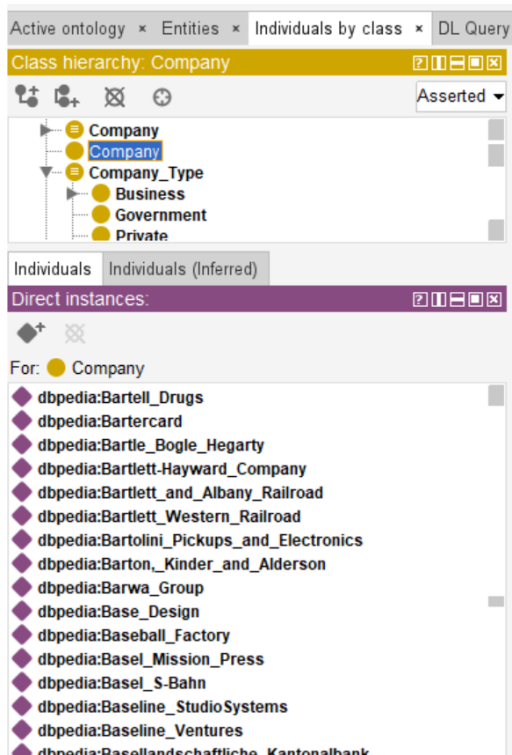
```

WHERE{
  ?company rdf:type dbpedia-owl:Company .
  ?company foaf:name ?name .

  OPTIONAL {?company dbpedia-owl:ceo ?ceo}
  OPTIONAL {?company dbpedia-owl:locationcity ?locationcity}
  OPTIONAL {?company dbpedia-owl:headquarter ?headquarter}
  OPTIONAL {?company dbpedia-owl:industry ?industry}
  OPTIONAL {?company dbpedia-owl:product ?product}
  OPTIONAL {?company dbpedia-owl:service ?service}
  OPTIONAL {?company dbpedia-owl:netIncome ?netIncome}
  OPTIONAL {?company dbpedia-owl:companyType ?companyType}
  OPTIONAL {?company dbpedia-owl:operatingIncome ?operatingIncome}
  OPTIONAL {?company dbpedia-owl:ranking ?ranking}
  OPTIONAL {?company dbpedia-owl:revenue ?revenue}
}
LIMIT 10000
"""

```

This figure above shows the construct query populating my ontology from dbpedia.



These screenshots above the show the instances of company data populated from dbpedia (left screenshot) and the CSV (right screenshot). More detail on the CSV data in **section 3**.

This sparql query is used to extract the data from external resource such as DBPedia and then populate into our ontology.

```
PREFIX company: <http://www.semanticweb.org/ontologies/user/Company#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?name ?country ?profit ?employees ?ranking
WHERE
{
  ?company rdf:type company:Company .
  ?company company:name ?name .
  ?company company:headquarter ?country .
  ?company company:profit ?profit .
  ?company company:number_of_employees ?employees .
  ?company company:ranking ?ranking .

  # Filter to include only companies headquartered in the United Kingdom
  FILTER(STR(?country) = "http://www.semanticweb.org/ontologies/user/Company#United_Kingdom")
}
LIMIT 5
"""
```

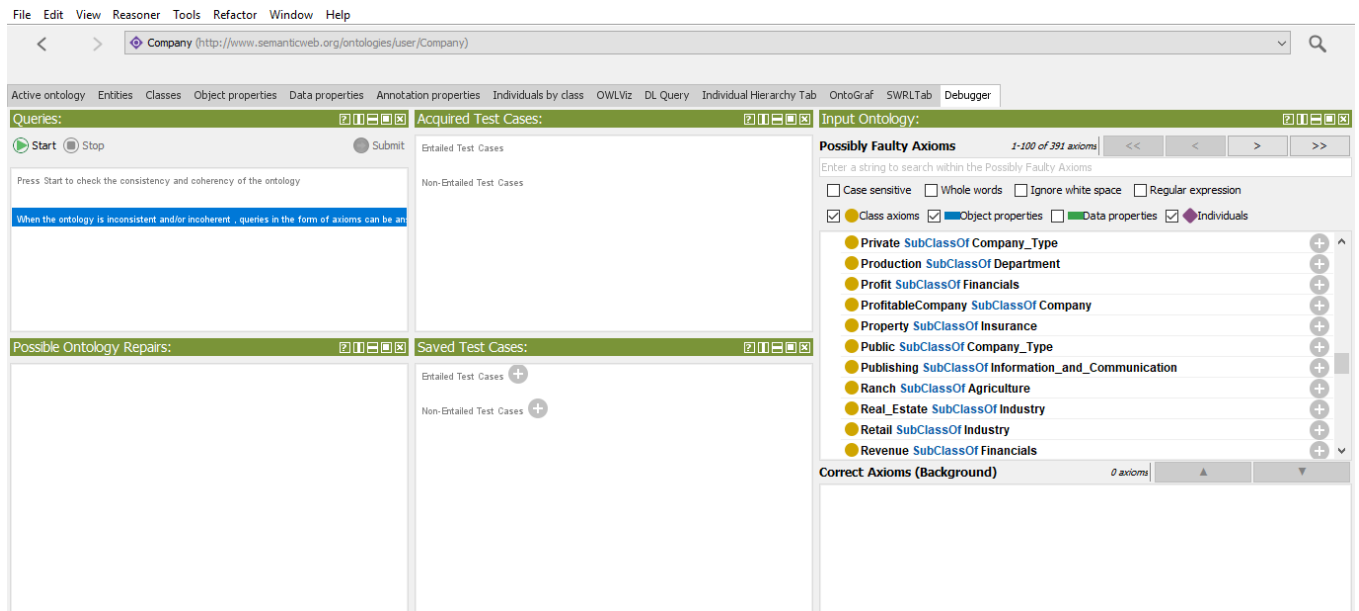
This sparql query is used to find the top countries of the United Kingdom with their employee strength and the profit details. Section 5 and 6 of this report cover general and advanced queries in further detail.

2.7 Justification, Explanation, and Validation of Ontological Modeling Decisions

This section provides the reasoning behind the modeling decisions. It also describes how the ontology was validated to ensure consistency:

- **Modeling Justification:** The decisions were based on standard business structures, logical relationships, and common ontology practices such as the departments in the company and the products they manufactured.
- **Validation:** The ontology was validated using Protégé's reasoner such as Pellet and Hermit to check for inconsistencies and ensure that all relationships were correctly inferred. For example, when we insert the SWRL rules pellet reasoner is used to check the consistency of the ontology.

In some cases, a few DL rules were not working due to issues in the queries format or the incorrect structure, so these issues were found using the pellet and Hermit reasoner and resolved.



3. Extra Task

3.1 Use of Appropriate, Diverse Data Sources

The data used to populate the ontology came from a variety of sources, ensuring diversity and relevance such as **DBPedia**, **csv dataset** from local storage and **sparql endpoint** is used:

- **Data Sources:** RDF dumps, SPARQL endpoints, and non-semantic datasets converted to RDF were used to populate the ontology.
- **Rationale:** The selection of these sources like **DBPedia** and **CSV Data** was based on the need to obtain comprehensive and reliable data for our ontology population.

```
sparql = SPARQLWrapper("http://dbpedia.org/sparql")
construct_query="""
    PREFIX org: <http://www.semanticweb.org/ontologies/user/Company.owl#>
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX foaf: <http://xmlns.com/foaf/0.1/>
    PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
    PREFIX dbpprop: <http://dbpedia.org/property/>
```

The csv dataset was obtained from linkedmdb (<https://data.world/datagov-uk/4462e41a-7413-4359-97f5-420b6ca5f9c0/workspace/file?filename=basic-company-data-part-6-3.zip>). After downloading the data, it was cleaned, removing null and unnecessary columns. The csv data included thousands of rows, a snapshot of the csv data is shown below.

A1												
	A	B	C	D	E	F	G	H	I	J	K	L
1	Id	CompanyName	CompanyNumber	RegAddress	RegAddress:PostTown	RegAddress	RegAddress	RegAddress	RegAddress	CompanyCategory	CompanyStatus	CountryOf Dissoluti
2	1	ROBO PUBLISHING LIMITED	3674253	21 MOUNT C	EDGWARE MIDDLESEX				HA8 9SY	Private Limited Company	Active	United Kingdom
3	2	ROBO RAIL LIMITED	7983452	SUITE 22 2N WHITEFRI	HARROW I MIDDLESE	UNITED KII	HA3 5RN			Private Limited Company	Active	United Kingdom
4	3	ROBO REPAIRS LIMITED	8550448	30 MAYFIELD CLOSE	FERNDOW DORSET				BH22 9HS	Private Limited Company	Active	United Kingdom
5	4	ROBO SHADOW LTD	10726476	92 FRIERN GARDENS	WICKFORLESSEX	ENGLAND	SS12 0HD			Private Limited Company	Active	United Kingdom
6	5	ROBO SHIPPING SERVICES LIMITED	12055946	387 HERRIES ROAD	SHEFFIELD	ENGLAND	SS 7HE			Private Limited Company	Active	United Kingdom
7	6	ROBO SM LTD	12237319	LLANERCH BARNFIELD	NEWTOWN	UNITED KII	SY16 2LA			Private Limited Company	Active	United Kingdom
8	7	ROBO SOFTWARE SYSTEMS LIMITED	2683434	THE IVY HO	1 FOLLY L PETERSFIE	HAMPSHIRE	ENGLAND	GU31 4AU		Private Limited Company	Active	United Kingdom
9	8	ROBO TECHNICS LIMITED	12155013	KEMP HOU	160 CITY R LONDON	UNITED KII	EC1V 2NX			Private Limited Company	Active	United Kingdom
10	9	ROBO TRADING LIMITED	11986846	10 MASONS HILL	LONDON	UNITED KII	SE18 6EJ			Private Limited Company	Active	United Kingdom
11	10	ROBO TRANSPORT LTD	12355988	32 FOLIAMB	NEW ROSS DONCASTER	UNITED KII	DN11 0SX			Private Limited Company	Active	United Kingdom
12	11	ROBO-PMO LTD	11446296	THE OLD RE	BARROWB GRANTHAM	UNITED KII	NG32 1BT			Private Limited Company	Active	United Kingdom
13	12	ROBO-TEST LTD	8944117	231B BUSIN	ISLINGTON LONDON	ENGLAND	N1 0QH			Private Limited Company	Active	United Kingdom
14	13	ROBO570 LTD	10243136	FIRST FLOO	125-135 PIBRIGHTON	ENGLAND	BN1 6AF			Private Limited Company	Active	United Kingdom
15	14	ROBOADVISORCOIN LTD	11721421	17 CARLISLE STREET	LONDON	UNITED KII	W1D 3BU			Private Limited Company	Active	United Kingdom
16	15	ROBOBACK LIMITED	12168975	1 WARDEN ABBEY	BEDFORD	UNITED KII	MK41 0SW			Private Limited Company	Active	United Kingdom
17	16	ROBOBITZ LTD	6551643	2 DAMER CL	MILTON A BLANDFOI	DORSET			DT11 0FB	Private Limited Company	Active	United Kingdom
18	17	ROBOBO LAND LIMITED	10895517	3 MERLIN C	GATEHOU: AYLESBURY	UNITED KII	HP19 8DP			Private Limited Company	Active	United Kingdom
19	18	ROBOBO MANAGEMENT LIMITED	9348755	3 CARRERA I	MERLIN C AYLESBUR	BUCKS	UNITED KII	HP19 8DP		Private Limited Company	Active	United Kingdom
20	19	ROBOBOND LIMITED	1796748	124/150 HA	LONDON				E2 7QS	Private Limited Company	Active	United Kingdom

3.2 Correct Mechanism to Retrieve and Transform Data to Fit Your Ontology

This section describes the process used to retrieve and transform data to fit the ontology structure:

- **Data Retrieval:** Data was retrieved from RDF dumps and SPARQL endpoints such as DBPedia and local file storage. The process involved connecting to these sources using the related libraries in python such as `rdflib.graph` , `SPARQLWrapper` , `csv` , `rdflib` , `rdflib.namespace` and extracting relevant information according to our ontology.
- **Data Transformation:** Non-semantic datasets such as csv data were converted to RDF to ensure compatibility with the ontology. This process involved mapping data to RDF triples and storing them in a compatible format. For this reason, we use the sparql queries to map the data to RDF tiples and stores in the compatible format in our ontology.

```

CONSTRUCT {
    ?company rdf:type org:Company .
    ?company org:name ?name .
    ?company org:ceo ?ceo .
    ?ceo rdf:type org:Ceos .
    ?company org:locationCity ?locationcity .
    ?locationcity rdf:type org:City .
    ?company org:headquarter ?headquarter .
    ?headquarter rdf:type org:Country .
    ?company org:industry ?industry .
    ?industry rdf:type org:Industry_Type .
    ?company org:product ?product .
    ?product rdf:type org:Main_Products .
    ?company org:service ?service .
    ?company org:netIncome ?profit .
    ?company org:companyType ?companyT .
    ?company org:numberOfEmployees ?number_of_employees .
    ?company org:operatingIncome ?operating_income .
    ?company org:operatingIncome ?operating_income .
    ?company org:ranking ?ranking .
    ?company org:revenue ?revenue .
}

```

3.3 Explanation of the Mechanism

The mechanism used to retrieve and transform data is explained in this section. It describes the steps involved and the tools used to achieve this:

- **Explanation of Data Mechanism:** The process involved connecting to SPARQL endpoints, extracting RDF data, and converting non-semantic data to RDF. For local file data extraction, in which data is non-format like CSV, I converted this data into RDF by creating RDF-triples using RDF library and assigning appropriate namespaces and types. This conversion ensures that all data is according to the consistent semantics structure.

```
# Define namespaces
org = Namespace("http://www.semanticweb.org/ontologies/user/Company.owl#")

# Read CSV data within the 'with' block
csv_file = "CompanyData.csv" # CSV file containing new data
with open(csv_file, "r", newline="", encoding="utf-8") as file:
    reader = csv.DictReader(file) # Dictionary reader for CSV

    for row in reader: # Iterate over rows within the 'with' block
        # Create unique URI for the company
        company_uri = URIRef("http://www.semanticweb.org/ontologies/user/Company#" + row["Id"])

        # Add RDF triples representing the new company
        existing_ontology.add((company_uri, RDF.type, org.Company))
        existing_ontology.add((company_uri, org.name, Literal(row["CompanyName"])))
        existing_ontology.add((company_uri, org.address, Literal(row["PostTown"])))
        existing_ontology.add((company_uri, org.company_type, Literal(row["CompanyCategory"])))
        existing_ontology.add((company_uri, org.company_Number, Literal(row["CompanyNumber"])))

# Serialize and save the updated ontology
existing_ontology.serialize(destination="company_basic.owl", format="xml")
```

- **Tools and Techniques:** The Python script was used for data retrieval and transformation, and the SPARQL queries were used to extract specific information. After retrieval and transform the data according to our ontology, I wrote the sparql queries using SQLWrapper library in python to populate the data into ontology and then serialized the updated ontology for storage in xml format so we can process it further.

Once the population and manual testing was completed, I tested the correctness of the ontology by running the few sparql queries and which convinced me that ontology is properly constructed, and data is represented in an appropriate way. When I loaded the data from DBPedia, I faced a few issues such as the data for few classes did not load, thus I solved this issue by analyzing the data and object structure, and making the correction in the object hierarchy and by adding more classes to hold the data and check the correctness of the characteristics of the object.

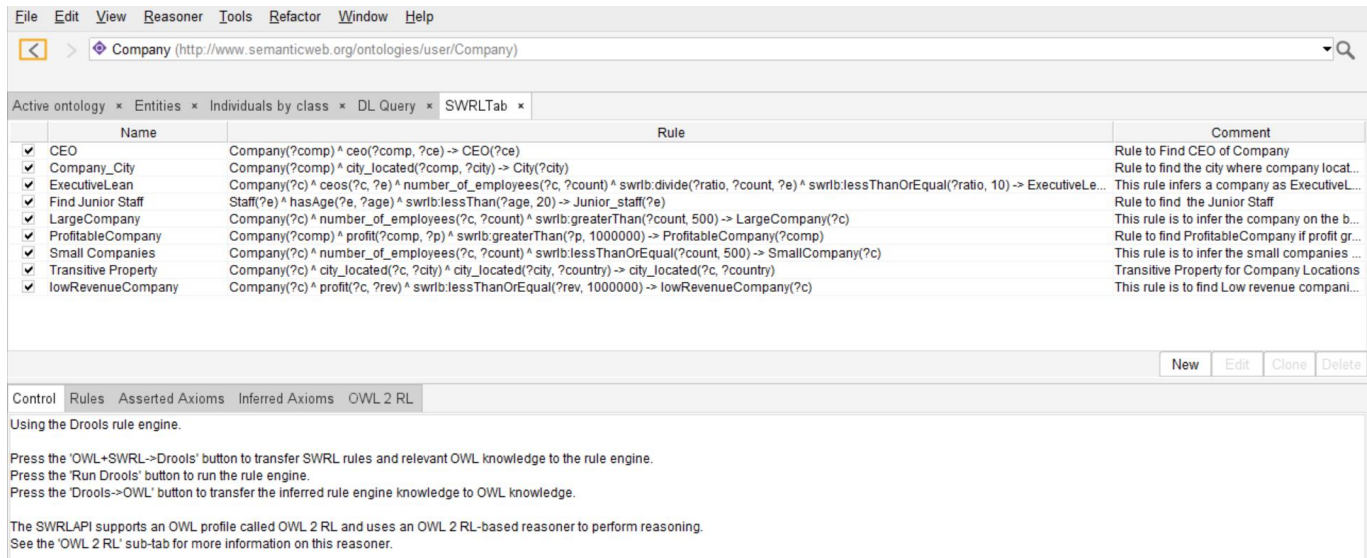
4. Advanced Task

4.1 Correctly Working (Inferencing) Ontology with Reasoner and SWRL Rules

This section discusses how the ontology uses a reasoner and SWRL rules to infer new relationships and ensure logical consistency:

- **Use of Reasoner:** The HermiT 1.3.8 and Pellet reasoner in Protégé was used to check for inconsistencies and infer relationships based on SWRL rules.
- **SWRL Rules:** Various SWRL rules were implemented to define logic-based inferences. For example, rules were used to classify types of revenue, or employees based on age.

Furthermore, rules are defined to classify the **company** based on city and the **CEO** of the company based on Company name.



Name	Rule	Comment
✓ CEO	<code>Company(?comp) ^ ceo(?comp, ?ce) -> CEO(?ce)</code>	Rule to Find CEO of Company
✓ Company_City	<code>Company(?comp) ^ city_located(?comp, ?city) -> City(?city)</code>	Rule to find the city where company locat...
✓ ExecutiveLean	<code>Company(?c) ^ ceos(?c, ?e) ^ number_of_employees(?c, ?count) ^ swrlb:divide(?ratio, ?count, ?e) ^ swrlb:lessThanOrEqual(?ratio, 10) -> ExecutiveLe...</code>	This rule infers a company as ExecutiveLe...
✓ Find Junior Staff	<code>Staff(?e) ^ hasAge(?e, ?age) ^ swrlb:lessThan(?age, 20) -> Junior_staff(?e)</code>	Rule to find the Junior Staff
✓ LargeCompany	<code>Company(?c) ^ number_of_employees(?c, ?count) ^ swrlb:greaterThan(?count, 500) -> LargeCompany(?c)</code>	This rule is to infer the company on the b...
✓ ProfitableCompany	<code>Company(?comp) ^ profit(?comp, ?p) ^ swrlb:greaterThan(?p, 1000000) -> ProfitableCompany(?comp)</code>	Rule to find ProfitableCompany if profit gr...
✓ Small Companies	<code>Company(?c) ^ number_of_employees(?c, ?count) ^ swrlb:lessThanOrEqual(?count, 500) -> SmallCompany(?c)</code>	This rule is to infer the small companies ...
✓ Transitive Property	<code>Company(?c) ^ city_located(?c, ?city) ^ city_located(?city, ?country) -> city_located(?c, ?country)</code>	Transitive Property for Company Locations
✓ lowRevenueCompany	<code>Company(?c) ^ profit(?c, ?rev) ^ swrlb:lessThanOrEqual(?rev, 1000000) -> lowRevenueCompany(?c)</code>	This rule is to find Low revenue compani...

Control Rules Asserted Axioms Inferred Axioms OWL 2 RL

Using the Drools rule engine.

Press the 'OWL+SWRL->Drools' button to transfer SWRL rules and relevant OWL knowledge to the rule engine.
 Press the 'Run Drools' button to run the rule engine.
 Press the 'Drools->OWL' button to transfer the inferred rule engine knowledge to OWL knowledge.

The SWRLAPI supports an OWL profile called OWL 2 RL and uses an OWL 2 RL-based reasoner to perform reasoning.
 See the 'OWL 2 RL' sub-tab for more information on this reasoner.

Here are some of the few SWRL rules that we implement in the protégé according to our ontology to infer the relationships and classes.

`Staff(?e) ^ hasAge(?e, ?age) ^ swrlb:lessThan(?age, 20) -> Junior_staff(?e)`

`Company(?c) ^ number_of_employees(?c, ?count) ^ swrlb:greaterThan(?count, 500) -> LargeCompany(?c)`

`Company(?comp) ^ profit(?comp, ?p) ^ swrlb:greaterThan(?p, 1000000) -> ProfitableCompany(?comp)`

`Company(?c) ^ profit(?c, ?rev) ^ swrlb:lessThanOrEqual(?rev, 1000000) -> lowRevenueCompany(?c)`

`Company(?c) ^ city_located(?c, ?city) ^ city_located(?city, ?country) -> city_located(?c, ?country)`

`Company(?c) ^ ceos(?c, ?e) ^ number_of_employees(?c, ?count) ^ swrlb:divide(?ratio, ?count, ?e) ^ swrlb:lessThanOrEqual(?ratio, 10) -> ExecutiveLeanCompany(?c)`

`Company(?c) ^ number_of_employees(?c, ?count) ^ swrlb:lessThanOrEqual(?count, 500) -> SmallCompany(?c)`

4.2 Correct A-Box with Enough Individuals

The A-Box in the ontology contains sufficient individuals to validate the defined logic rules. This section explains how the individuals are structured and how relationships are inferred:

- **Individuals in the A-Box:** The A-Box contains a variety of instances representing companies, employees, products, Person details, Countries, and city details etc.
- **Inferences from the A-Box:** Most relationships are inferred through the reasoner, ensuring that hardcoded values are minimized to maintain flexibility and adaptability.

Here is the list of individuals we asserted to our ontology according to classes.

Class	Individuals
Company	Airbus, Amzaon.com, Apple, Boeing, Boeing_Uk,Aviva_Insurance, Microsoft, Prosche, Toyota_Group, Walmart
Industry	Aerospace, Banking, Hospitality, Insurance, Retail, Telecommunication, Gas, Builder, Information_Technology
Products	Aeroplanes, Automobiles, Computers, Electricity, Petrol, Software,
Service	Banking, Communication, Construction, Hospitality Service, Information_Service
Locations	Beijing, California, Dallas, Leiden, London, Seattle, China, USA, Netherlands
Person	Tim_Cook, Leo_Quinn, Daniel_Zhang, Amanda_Blanc, Bernard_Lonney, Jes_Staley, Nuno_Metos

4.3 Correctly Commented Rules and Explanations in the Report

This section discusses the importance of comments and explanations for SWRL rules and provides a brief explanation of how the rules are used:

- **Commented SWRL Rules:** Each SWRL rule includes comments to explain its logic and expected outcomes. This ensures that the rules are understandable and maintainable.

Name	CEO
Comment	Rule to Find CEO of Company
Status	Ok
SWRL Rule	Company(?comp) ^ ceo(?comp, ?ce) -> CEO(?ce)

In this SWRL rule, comment was added to show that this rule is used to find the CEO of the company.

In this SWRL rule, comments are added to show that finding the city where the company is located. Similarly, I added comments for the rest of the SWRL rules.

5. Sparql Queries

To test the ontology, I ran multiple queries to extract the information like headquarters of the company, products manufactured by the companies and the profits generated by the company. Here are some examples.

Query to find the ?name ?ranking

PREFIX company: <<http://www.semanticweb.org/ontologies/CourseWork/Company#>>

PREFIX rdf: <<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>

For ranking, and company names

SELECT DISTINCT ?name ?ranking

WHERE

```
{
    ?company rdf:type company:Company .
    ?company company:name ?name .
    ?company company:ranking ?ranking .
}
```

Query to find and filter the ?name ?revenue greater than 10000000

PREFIX org: <http://www.semanticweb.org/ontologies/user/Company#>

PREFIX rdf: <<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>

SELECT ?company ?name ?revenue

```

WHERE {
    ?company rdf:type org:Company .
    ?company org:name ?name .
    ?company org:revenue ?revenue .
    FILTER (?revenue > 1000000) # Modify as needed
}

```

Query to find Company, name, ceo, location and headquarter.

```

#Company, name, ceo, location and headquarter
PREFIX org: <http://www.semanticweb.org/ontologies/user/Company#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?company ?name ?ceo ?locationcity ?headquarter
WHERE {
    ?company rdf:type org:Company .
    ?company org:name ?name .
    OPTIONAL { ?company org:ceo ?ceo }
    OPTIONAL { ?company org:city_located ?locationcity }
    OPTIONAL { ?company org:headquarter ?headquarter }
}

```

Query to find Company, address, company Type and company Number by applying filter on company Number.

```

PREFIX org: <http://www.semanticweb.org/ontologies/user/Company.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?company ?name ?address ?companyType ?companyNumber
WHERE {
    ?company rdf:type org:Company .
    ?company org:name ?name .
    ?company org:address ?address .
    ?company org:company_type ?companyType .
    ?company org:company_Number ?companyNumber .
    FILTER(STR(?companyNumber) = "04433030") # Company number to match
}

```

6. Advanced Queries

Query to find UK Companies with Largest Profit Per Employee (complete code can be found in python file)

```
org = Namespace("http://www.semanticweb.org/ontologies/user/Company#")

query = """
PREFIX org: <http://www.semanticweb.org/ontologies/user/Company#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?company ?name ?profit ?employees
WHERE {
  ?company rdf:type org:Company .
  ?company org:name ?name .
  ?company org:headquarter ?country .
  ?company org:profit ?profit .
  ?company org:number_of_employees ?employees .

  # Filter to include only companies headquartered in the United Kingdom
  FILTER(STR(?country) = "http://www.semanticweb.org/ontologies/user/Company#United_Kingdom")
}
"""
```

UK Companies with Largest Profit per Employee:

Company Name	Profit	Employees	Profit per Employee
British Petroleum	10040000000.00	22000	456363.64
HSBC	23670000000.00	85000	278470.59
Aviva Insurance	16100000000.00	31000	51935.48
Barclays	10200000000.00	83500	12215.57

Query for companies and employees with >= 3 locations

```
# Define the SPARQL query to fetch company names with 3 or more locations
query = """
PREFIX org: <http://www.semanticweb.org/ontologies/user/Company#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?company ?name ?employees ?locations
WHERE {
  ?company rdf:type org:Company .
  ?company org:name ?name .
  ?company org:number_of_employees ?employees .
  ?company org:number_of_locations ?locations .
  FILTER(xsd:integer(?locations) >= 3)
}
"""
```

Query for companies and employees with >= 3 locations!

Company	Name	Employees	Locations
Airbus	Airbus	133671	3

7. Issues and Resolutions

During the implementation of my company ontology, I ran into the following issues. There were four to five reasoners which I attempted using. However, only pellet and HermiT 1.3.8 worked and in old versions different rules were not working, or some reasoners did not support DL rules or Object Property assertions.

Although Pallet has the best performance out of the five reasoners, it also has problems. One main problem I found while building the ontology is SWRL rule does not have built in atom for different operations like intersection, union. For example, if I wanted to implement this rule for

the industry class, it results in error, rules like these were not implementable for me in protégé.

Company_Type \cap (industry_type_companies \cup Business)

To solve this problem, I adopted the new approach to assign this rule to company type class and use the property that is executable and correctly inferred.

Company and (address some Country)

Products and (associated_with some Main_Products)

Company_Type and (industry_type_companies some Business)

These rules are supported by the pellet and HermiT 1.3.8 reasoner and correctly inferred the results instead of using the Intersection or union symbols.

Also, when writing the description logic using this approach, it gave the error that this operator is not found in protégé or syntax error. The Pellet reasoner did not work well for the SWRL rules that I defined for the transitive property to the Data property **city_located using DL rule syntax**, it threw error. Thus, I used the SWRL approach for assigning and declaring the DL rule and it correctly works

```
Company(?c) ^ city_located(?c, ?city) ^ city_located(?city, ?country) -> city_located(?c, ?country)
```

Furthermore, the main issue is not all relevant data populated into the ontology that leading to knowledge representation issues. To resolve this problem, I reviewed the data source such as DBPedia and python population scripts to ensure that all necessary data is being captured and transformed into RDF format. So, I adjusted the population process as needed to fill the gaps in the ontology. When writing the sparql queries for populating and extracting the data from the ontology, I faced the issue of errors in the sparql queries. So, I carefully reviewed the scripts and sparql queries for syntax errors and inconsistencies to extract and populate the data efficiently.

Another issue that I encountered was improperly defined domain and range restrictions for properties that can lead to incorrect inferences or data inconsistencies. For example, I assigned the wrong domain and range to **city_located** and **industry_type** data property. I ran the reasoner and sparql queries to find the results and when wrong data extracted, I carefully reviewed and resolved this by assigning the correct domain and range according to our ontology design and results. Assigning properties incorrectly to classes also results in incorrect inferences or inconsistencies. So, I double checked all the property assignments for each class to ensure that it is accurately reflecting the intended relationships.

8. Conclusion

To conclude, the company ontology was designed to meet the requirements of the assignment. The ontology is a well-structured taxonomy, property hierarchy, with correct domain and range restrictions. Using SWRL rules and the Pellet reasoner, the ontology can infer relationships and ensure logical consistency throughout the ontology. Diverse data sources such as DBPedia, local CSV data and Sparql queries were used to populate and test the ontology which led to constant refinement of my designs. This report provides a complete overview of the ontology design, implementation, and validation.

9. List of Files

1. Project Report PDF for the company ontology.
2. Company.owl Ontology file of the company.
3. Company_basic.owl is the populated ontology file attached incase.
4. CompanyData.csv for the second data source (local csv).
5. Populate_ontology.py script to populate the data from the DBPedia, results in the populated file called company_basic.owl being outputted.
6. Query.py python script to execute the test queries and see the results.