Big Data Research Project

Emir Nurmatbekov
Anant Gupta

Geology Knowledge Base Construction

Supervisors:	Sylvain Wlodarczyk	Schlumberger	SWlodarczyk@slb.com
	Nacéra Seghouani	CentraleSupelec	nacera.seghouani@centralesupelec.fr
	Francesca Bugiotti	Schlumberger	francesca.bugiotti@centralesupelec.fr

Contents

CHAPTER 1

Introduction

1.1 General context

The previous couple of decades have demonstrated the burgeoning of data and knowledge, in both size and diversity in all industries.Continuous evolution of the computational power and technology has complemented the approach in which the data is processed to extract valuable information. Knowledge bases are a prime evidence to the above statement. They have become ubiquitous in every field, be it Wikipedia's displaying results or Google's omnipotent search engine. Thanks to the blossoming growth of technology giants: Google, Microsoft, Facebook, Amazon, more and more companies aim to capitalize the reasoning and inferential power of these knowledge bases.

The stakeholders, Schlumberger, aspire to build a system, capable enough of answering question strictly pertinent to stratigraphy in the north-sea. The sole intent is research and devise techniques which help in automatically provide answers to user's questions along with having the aptitude of evolving with time.

The proposed paradigm should be to answer questions like:

- Where is Ekofisk Formation?
- What are the wells crossing Ekofisk Formation?
- What is the group of the Ekofisk Formation?
- What is the top depth of the Ekofisk Formation for the well 1/2-14?
- What are the members of Ekofisk formation?
- What is the period and age of Ekofisk formation?

1.2 The Problem

It is true that knowledge bases have been created earlier in the field of geology but they have either been created by single team at company/ university, tailored from one point of view. The proposed methods severely depend on the in-house data and

later fail accommodate data from other sources due varying column names or data shapes.
Secondly, the ontology in most use cases should not be designed as a static behaviour of the domain and should have the ability to guarantee the natural evolution of its domain. Neglecting this automation of the evolution process would fail the adoption of ontologies for several use cases, since we would only be developing yet another static knowledge representation, yet another standard. In other words, the ontology has to evolve in either automatic or semi-automatic way.
As explained in the later sections, our unfruitful approaches: traditional knowledge base and virtual integration of sources, the problems statement was not being exactly solved. For example, the traditional approach demanded the manual creation of the TBox and ABox, along with their inter-connections. An evolution, triggered either due to a change in the data source or change in ontology, meant doing the entire process all over again. Similarly for the virtual integration with every incoming data source, the mappings from source to ontologies required to be mapped manually.
The objective is to create a knowledge base and a system that would allow it to grow and evolve with new different sources coming in.

1.3 Our Solution

The presented problem statement is quite contemporary and is open to interpretations, making it a perfect foil for a research topic. We envisioned this challenge to try a couple of different techniques, understanding the value addition and shortcomings of each.

We propose an approach where new sources are automatically added to the ontology without manual intervention. Automatic addition is not just appending everything to the ontology but examining a source and trying to locate its match in the existing ontology. The entire identification process is coded and takes in as input the existing ontology and the incoming data source. The function does first distinguishes the class and checks for a similar presence in the ontology. Next, we identify the properties of the classes and check for the presence of a similar property inside the ontology. Additionally, a check is performed to validate if the property already linking the same classes. At last, we determine the range and the domain for every property where ranges are treated as type 'Literals'. The output of the process is a set of new yet non-redundant mappings since not all concepts are blindly added to the system.

1.4 Summary of the following sections

The succeeding content of the report we first, discuss in brief the state-of-art techniques being used to automatically generate ontologies since this automation was a basic yet crucial feature for the success of a proposal. Second, we move to the implementation different techniques: Traditional and Virtual Mapping. Through the summary of these techniques we establish the pro-cons of each, which later act as building blocks to our idea. Next , we move the important segment of our proposed methodology. The data collection process is discussed to give the reader a fair sense idea of volume and variety of sources involved in the evolution of the knowledge graph. Later, we throw light on the mappings initiate by the classification of classes, properties along with their range and domain. The similarity measure check of classes is discussed in detail to establish that the evolution is not mere addition of concepts. The methodology is concluded by giving an idea on how the queries will executed by the system. The experiments undertaken to illustrate our design, follow the methodology and is later terminated by the conlusion.

Next, we move to delving into the details of our proposed methodology and it's working.

Chapter 2

Related Work

In this chapter, we will briefly discuss some of the existing state-of-the-art tools and approaches for automatic ontology generation.

2.1 DB2OWl

DB2OWL is a tool for automatic database-to-ontology mapping. The tool creates ontology from a relational database. It looks for some particular cases of database tables to determine which ontology component has to be created from which database component. The mapping process starts by detecting some particular cases for tables in the database schema. According to these cases, each database component (table, column, constraint) is then converted to a corresponding ontology component (class, property, relation). The set of correspondences between database components and ontology components is conserved as the mapping result to be used later [Cullot 2007].

2.2 TANGO

TANGO (Table ANalysis for Generating Ontologies) is an approach to generating ontologies based on table analysis. TANGO attempts to

1. understand a table's structure and conceptual content;
2. discover the constraints that hold between concepts extracted from the table;
3. match the recognized concepts with ones from a more general specification of related concepts;
4. merge the resulting structure with other similar knowledge representations.

TANGO is thus a formalized method of processing the format and content of tables that can serve to incrementally build a relevant reusable conceptual ontology. [Tijerino 2005]

2.3 Data-driven ontology construction and evolution

The authors propose a comprehensive ontology generation and evolution method. The proposed method consists of the four following phases: data extraction, ontology construction, ontology connection, and ontology evolution. In the first phase, data from various sources is mapped to data entities to form a unified data structure. In the second phase, an initial ontology is generated via instance- driven ontology construction. In the third phase, the initial ontologies are organised in terms of the dimensions of the various business elements, such as machine, product, process, and scenarios. In the final phase, rules regarding ontology restrictions are formulated to realise ontology evolution that respond to changes in the manufacturing environment. [Huang 2019]

CHAPTER 3

Previous approaches

3.1 Introduction

After defining the problem statement, we strategised and come up two approaches: the orthodox way of creating a knowledge base and the virtual integration. The methodologies failed to address and solve main challenges something which we uncovered in our discussion with the stakeholders. However, it did leave us with crucial insights which we were able to incorporate in our third and final approach. The motive behind each along with the undertaken steps and the learning's in form of advantages and disadvantages can be found in the succeeding paragraphs.

3.2 Traditional approach

Since the objective was to build a knowledge base for stratigraphy, the first thing to come to our mind was the traditional knowledge base system. The motivation being it quite flexible in terms of the resources/ tools that can be used and it is hassle-free process with clear documentation.

It involves two layers: TBox(conceptual) and ABox(instance). First, the ontology, that is, TBox is created with all of the entities and properties from the data model. Then, the ABox is populated with all of the instances of the ontology entities. Finally,the TBox is connected with the ABox. Once they are connected, the knowledge graph is ready for use and can be queried using SPARQL.

The process is kick-started by identifying classes and literals. These were used to generate the TBox using Protégé, which emits an .owl file. The motivation behind Protégé being

The ABox population was implemented using a python library called 'rdflib' by creating triples (subject-predicate-object) from the data sources. The motivation was since entire code was in python and it open-source provided good support to undertake this mission.

Finally, to connect the generated TBox and ABox files, we made of GraphDB by OntoText . GraphDB[gra 2015] is a family of highly efficient, robust, and scalable RDF databases. It facilitates the load and use of linked data cloud datasets, as well as personal resources. By implementation of RDF4J framework interfaces, the W3C SPARQL Protocol specification it enables for ease of access and compatibility with the industry standards. Additionally it supports all RDF serialization formats.

Advantages of this approach:

- Freedom. There is no restriction on the usage of tools. Any tool can be chosen as per need of the hour.
- Easy. There is enough clear documentation to formulate and implement this approach.

Disadvantages of using :

- No evolution. Changes to an existing ontology is very cumbersome process and it more often requires everything to built from scratch again.
- Manual. The ontology(TBox) along with the querying needs manual intervention be executed.

3.3 Virtual Knowledge Graph

As our second approach, we implemented the modern Virtual Knowledge Graphs. The Virtual Knowledge Graph (VKG) approach, also known in the literature as Ontology-Based Data Access (OBDA), has become a popular paradigm for accessing and integrating data sources. In such approach, the data sources, which are normally relational databases, are virtualized through a mapping and an ontology, and presented as a unified knowledge graph, which can be queried by end-users using a vocabulary they are familiar with. At query time, a VKG system translates user queries over the ontology to SQL queries over the database system [et al. 2020]. For this project we chose VKG tool called Ontop. Although, there are several other VKG systems(Morph, SparqlMap), we had decided to go with Ontop since we believed it is more mature and popular compared to its competitors. The detailed comparison and benchmark of the VKG tools was done in [Milos Chaloupka 2020].

However, the chosen approach didn't bring us to the final goal. Even though Virtual Knowledge Graphs have their benefits such as:

- It saves space by keeping the data in the data sources and not loading them into the knowledge base;
- Materialisable. At any point, the user can materialise virtual knowledge graph;
- On-the-fly querying. SPARQL queries are translated into SQL on-the-fly, and data sources are queried during runtime.

The disadvantages of the Virtual Knowledge Base approach were more dramatic which led to the decision to abandon it:

- Manual. Ontop mappings have to be created manually which means every time there is a change in the data sources whether it is new data source or change in the existing ones, the user has to write the mappings again.
- Tool-locked. The only way to implement the project via this approach was through Protege [Team 2015] or Ontop [et al. 2020].
- Not possible to integrate. None of the tools have working APIs, so we could not integrate it with the new approach.

Chapter 4
Methodology

4.1 Introduction

After reaching a dead-end with the previous two approaches, we proposed a new approach which takes into account the learning's of the previous ones and provides a solution to the problem at hand. Unlike the previous approaches, this approach, allows the possibility for ontology evolution and this process is validated in the succeeding section of demonstration.
We start with a very basic ontology model which includes the central objects of stratigraphy such as groups, formations, member and wellbores. Later, with arrival of new sources, the ontology is able to adapt and change its structure, add new concepts and properties along with latest instances to the existing classes. In the following sections, we will delve into greater depth of each of the step undertaken in the approach.

4.2 Data collection

For the MVP(prototype) the idea was to demonstrate a system's capacity to ingest and accept data from varied sources. The system should have the potential to find similar concepts and properties but at the same time not sacrifice the ability to evolve the ontology.

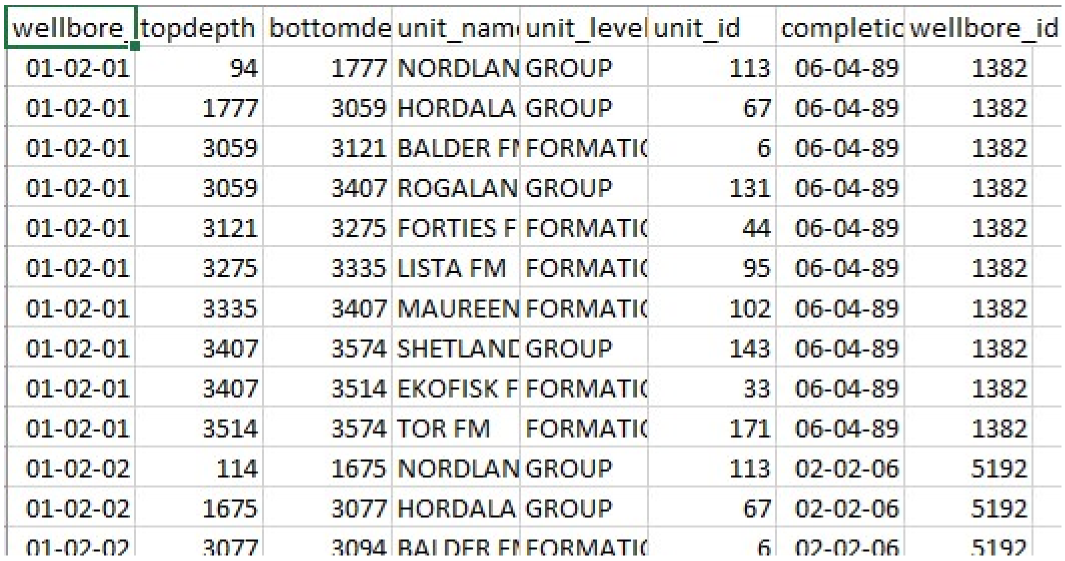
Since the stakeholders are interested only in the stratigraphy questions in and around the North-sea, we utilize the most dependent resource: Norwegian Petroleum Directorate(NPD) and its factpages. NPD[1] is a governmental specialist directorate and administrative body, having their primary objective as to contribute to the greatest possible values from the oil and gas activities to the Norwegian society, through efficient and responsible resource management. Along with this resource we make use of Wikipedia and Norlex group for scrapping data.

To keep the process of data ingestion fairly synonymous to the real world, we do not alter the data much except just removing the non-pertinent columns. For example, each website stores their information using an unique ID. It does not make sense to store these ID's since firstly the ID's will vary across sources and secondly, the concept name is quite potent since it is uniquely identifiable.

[1]https://www.npd.no/en/about-us/

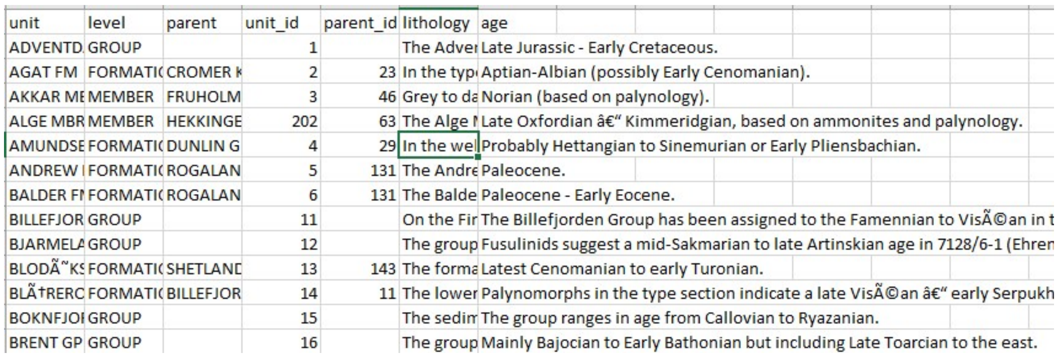
So to mimic the real world evolution, we make use of 8 data files in the form of CSV's. Each file varied on the number of rows and columns. A few notable files are follows:

- Wellbores
  These form a part of the initial ontology and contains all the details about the wellbores.
  Row count = 38,000



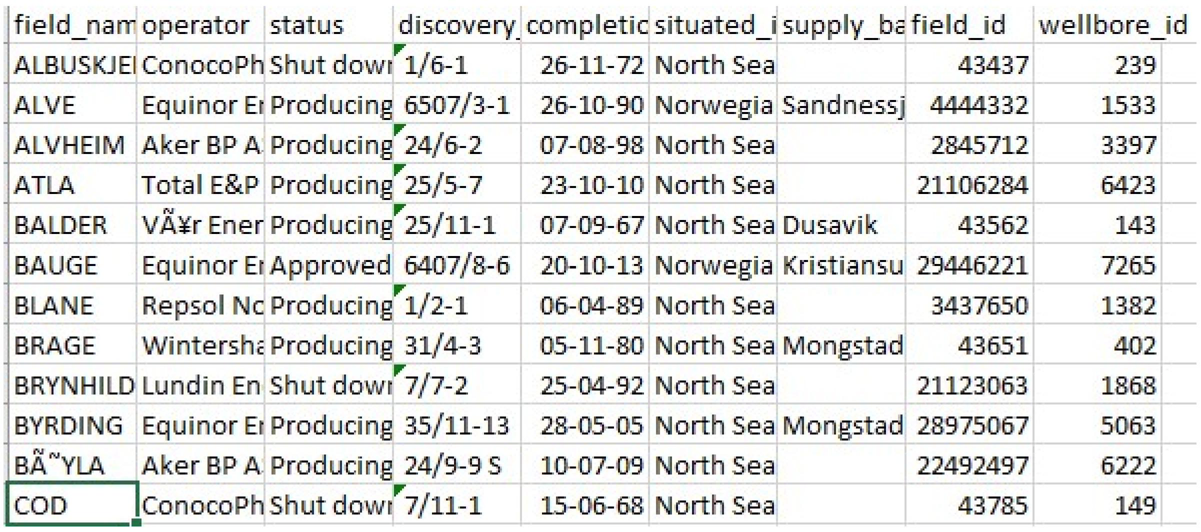
wellbore_	topdepth	bottomde	unit_nam	unit_level	unit_id	completic	wellbore_id
01-02-01	94	1777	NORDLAN	GROUP	113	06-04-89	1382
01-02-01	1777	3059	HORDALA	GROUP	67	06-04-89	1382
01-02-01	3059	3121	BALDER FI	FORMATI(	6	06-04-89	1382
01-02-01	3059	3407	ROGALAN	GROUP	131	06-04-89	1382
01-02-01	3121	3275	FORTIES F	FORMATI(	44	06-04-89	1382
01-02-01	3275	3335	LISTA FM	FORMATI(	95	06-04-89	1382
01-02-01	3335	3407	MAUREEN	FORMATI(	102	06-04-89	1382
01-02-01	3407	3574	SHETLANE	GROUP	143	06-04-89	1382
01-02-01	3407	3514	EKOFISK F	FORMATI(	33	06-04-89	1382
01-02-01	3514	3574	TOR FM	FORMATI(	171	06-04-89	1382
01-02-02	114	1675	NORDLAN	GROUP	113	02-02-06	5192
01-02-02	1675	3077	HORDALA	GROUP	67	02-02-06	5192
01-02-02	3077	3094	BALDER FI	FORMATI(	6	02-02-06	5192

- Formations
  These are retrieved from the 'lithostrat_unit.csv' and contain all information related to the formations.
  Row count = 160



unit	level	parent	unit_id	parent_id	lithology	age
ADVENTD.	GROUP		1		The Adver	Late Jurassic - Early Cretaceous.
AGAT FM	FORMATI(	CROMER K	2	23	In the typ	Aptian-Albian (possibly Early Cenomanian).
AKKAR ME	MEMBER	FRUHOLM	3	46	Grey to da	Norian (based on palynology).
ALGE MBR	MEMBER	HEKKINGE	202	63	The Alge I	Late Oxfordian â€“ Kimmeridgian, based on ammonites and palynology.
AMUNDSE	FORMATI(	DUNLIN G	4	29	In the wel	Probably Hettangian to Sinemurian or Early Pliensbachian.
ANDREW I	FORMATI(	ROGALAN	5	131	The Andre	Paleocene.
BALDER FI	FORMATI(	ROGALAN	6	131	The Balde	Paleocene - Early Eocene.
BILLEFJOR	GROUP		11		On the Fir	The Billefjorden Group has been assigned to the Famennian to VisÃ©an in t
BJARMELA	GROUP		12		The group	Fusulinids suggest a mid-Sakmarian to late Artinskian age in 7128/6-1 (Ehren
BLODÃ˜KS	FORMATI(	SHETLANE	13	143	The forma	Latest Cenomanian to early Turonian.
BLÃ†RERC	FORMATI(	BILLEFJOR	14	11	The lower	Palynomorphs in the type section indicate a late VisÃ©an â€“ early Serpukh
BOKNFJOI	GROUP		15		The sedim	The group ranges in age from Callovian to Ryazanian.
BRENT GP	GROUP		16		The group	Mainly Bajocian to Early Bathonian but including Late Toarcian to the east.

- Field
  It is initially not a part of the base ontology and is used to evolve the ontology.
  Row count = 150



field_nam	operator	status	discovery_	completic	situated_i	supply_ba	field_id	wellbore_id
ALBUSKJEI	ConocoPh	Shut dowı	1/6-1	26-11-72	North Sea		43437	239
ALVE	Equinor Eı	Producing	6507/3-1	26-10-90	Norwegia	Sandnessj	4444332	1533
ALVHEIM	Aker BP A	Producing	24/6-2	07-08-98	North Sea		2845712	3397
ATLA	Total E&P	Producing	25/5-7	23-10-10	North Sea		21106284	6423
BALDER	VÃ¥r Ener	Producing	25/11-1	07-09-67	North Sea	Dusavik	43562	143
BAUGE	Equinor Eı	Approved	6407/8-6	20-10-13	Norwegia	Kristiansu	29446221	7265
BLANE	Repsol Nc	Producing	1/2-1	06-04-89	North Sea		3437650	1382
BRAGE	Wintershı	Producing	31/4-3	05-11-80	North Sea	Mongstad	43651	402
BRYNHILD	Lundin En	Shut dowı	7/7-2	25-04-92	North Sea		21123063	1868
BYRDING	Equinor Eı	Producing	35/11-13	28-05-05	North Sea	Mongstad	28975067	5063
BÃ˜YLA	Aker BP A	Producing	24/9-9 S	10-07-09	North Sea		22492497	6222
COD	ConocoPh	Shut dowı	7/11-1	15-06-68	North Sea		43785	149

4.3 Base model

As mentioned in the previous section, our base model includes only the central concepts of the field of stratigraphy, and leaves room for the ontology to evolve and identify the rest of the concepts by itself from the data sources. The main concepts of stratigraphy are groups, formations, members and wellbores. For lithostratigraphic units: group, formation, member, we define a parent class `Unit` as `Owl:Class`, as well as we define each one of them as `rdfs:subClassOf Unit`. Wellbores are also represented as an `Owl:Class` called `Wellbore` which has a subclass `Core`. The class `Core` represents all the wellbores which have cores. The full schema of the base ontology is shown on Figure 4.1.

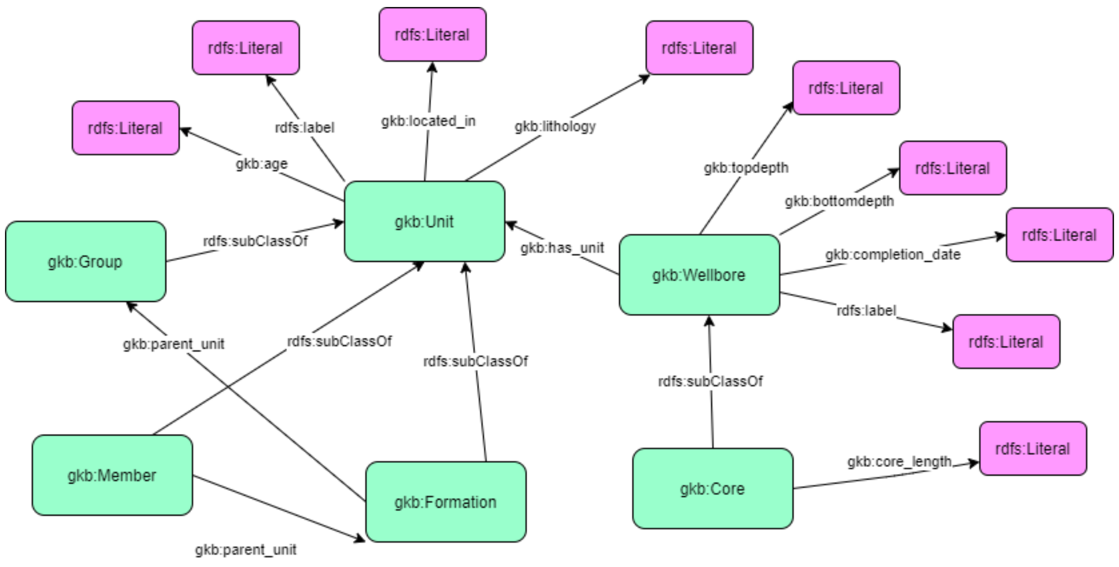


Figure 4.1: Base ontology

To make the base ontology schema more clear, below we describe each of the classes and properties of the ontology.

Classes:

- `Unit` - the main class which comprises of classes: `Group`, `Formation`, `Member`. Represents all lithostratigraphic units;
- `Group` - subclass of `Unit`. Represents groups;
- `Formation` - subclass of `Unit`. Represents formations;
- `Member` - subclass of `Unit`. Represents members;
- `Wellbore` - another main class, has a subclass `Core`. Represents all wellbores;
- `Core` - subclass of `Wellbore`. Represents wellbores with cores.

Properties:

- `age` - represents the age of the corresponding `Unit`;
- `located_in` - represents the location, area of the corresponding `Unit` or `Wellbore`;
- `lithology` - represents the lithology of the corresponding `Unit`;
- `has_parent` - represents the parent unit of the corresponding `Unit`.
- `has_unit` - represents the `Unit` that the corresponding `Wellbore` crosses.

- `topdepth` - represents the top depth of the corresponding `Wellbore`;
- `bottomdepth` - represents the bottom depth of the corresponding `Wellbore`;
- `topdepth` - represents the top depth of the corresponding `Wellbore`;
- `completion_date` - represents the completion date of the corresponding `Wellbore`;
- `core_length` - represents the core length of the corresponding `Core`.

4.4 Evolution

This is the part where we explain our main contribution to this project. The main challenge and the objective of the projects is to create an approach where the knowledge base evolves in an automated or semi-automated way. As we have stated in 1.2, our previous approaches were manual and required re-modelling of the ontology with changes in the data sources. Below, we will explain our ideas and new methodology which helped us to achieve the evolution of the ontology based on new data sources.
With a new source coming in, we need to ask ourselves several questions:

- What is this source?
- Does it bring new concepts?
- Does it just supplement the existing concepts with new instances?
- What are the properties?

The system developed in our approach answers these questions by itself without the user being involved and based on the answers modifies the ontology. The high level view of the system is shown on Figure 4.2.

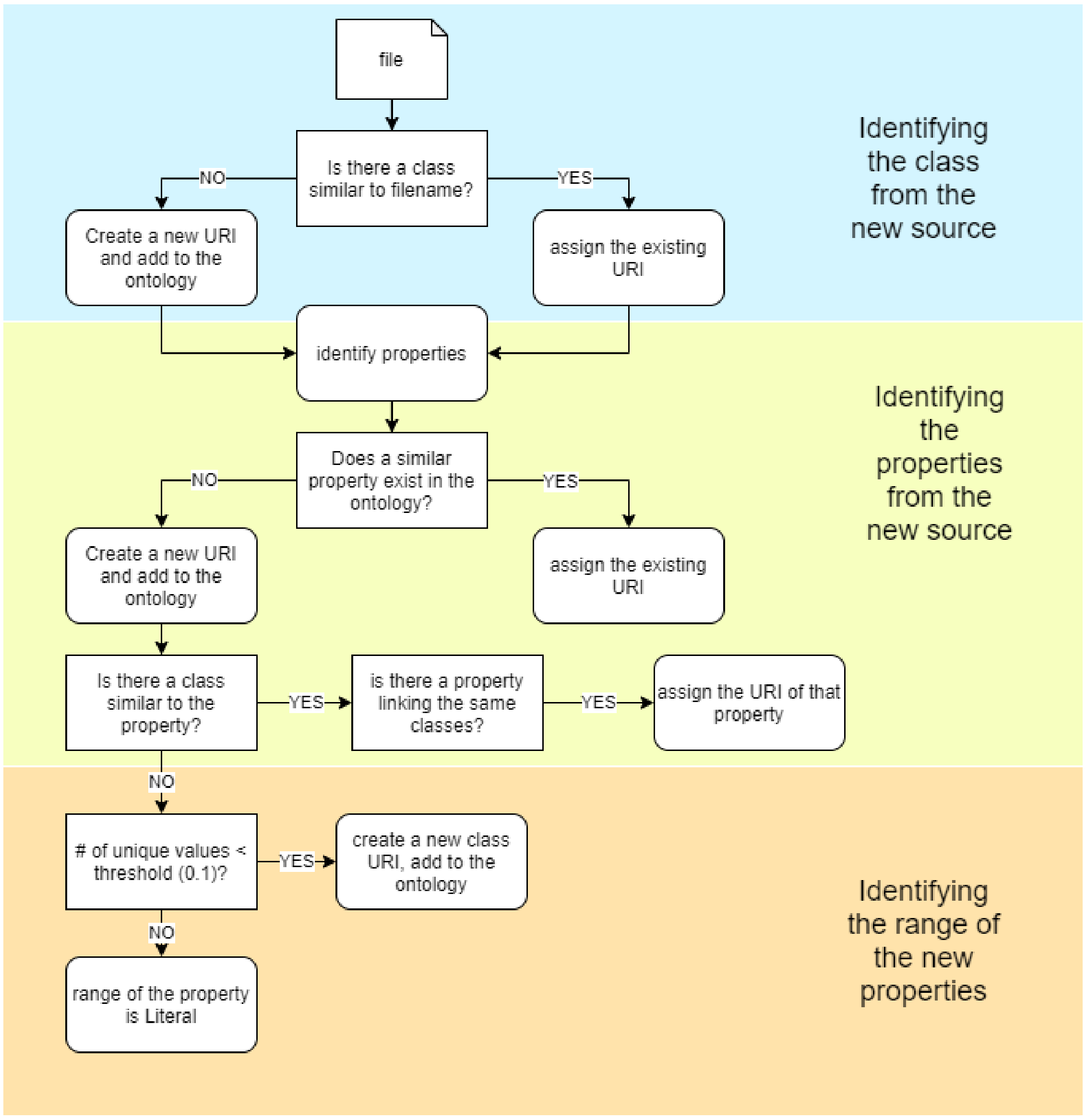

file
Is there a class similar to filename?
NO
YES
Create a new URI and add to the ontology
assign the existing URI
Identifying the class from the new source
identify properties
Does a similar property exist in the ontology?
NO
YES
Create a new URI and add to the ontology
assign the existing URI
Identifying the properties from the new source
Is there a class similar to the property?
YES
is there a property linking the same classes?
YES
assign the URI of that property
NO
# of unique values < threshold (0.1)?
YES
create a new class URI, add to the ontology
NO
range of the property is Literal
Identifying the range of the new properties

Figure 4.2: High level view of the ontology evolution system

4.4.1 Identifying classes

As shown on Figure 4.2, the first phase is identifying the class from the new source. Depending on whether the class already exists or not, it will assign the appropriate URI to the new class. The existence of the class is checked by the function `closest()`. The function consists of two parts. The first part checks if the class is similar to any existing classes syntactically meaning it will check if the class name is contained in the names of any other classes or if they have common substrings (refer to algorithm 1). If no syntactically similar class was found, then the function checks if any of the existing classes have semantical similarity with the new class. Precisely, it checks if any of the words in classes have common synonyms with the words in the new class (refer to algorithm 2). For full code refer to our github repository.

Algorithm 1 closest() function algorithm. Part 1. Syntactic similarity

```
Input: candidate, types
closest_types ← [ ]
for type in types do
    if type.contains(candidate) then
        cand_in_type ← True
    subtypes ← type.split("_")
    type_in_cand ← 0
    for subtype in subtypes do
        if candidate.contains(subtype) then
            type_in_cand ← type_in_cand + 1
    if cand_in_type is True || type_in_cand > 0 then
        closest_types ← (type, type_in_cand)
if closest_types not empty then
    closest_type ← closest_types[argmax(closest_types)]
return closest_type
```

Algorithm 2 closest() function algorithm. Part 2. Semantic similarity

```
Input: candidate, types
for type in types do
    subtypes ← type.split("_")
    cands ← candidate.split("_")
    syns ← 0
    for c in cands do
        for subtype in subtypes do
            if common_syns(subtype, c) is True then
                syns ← syns + 1
    if syns >= len(subtypes) || syns >= len(cands) then
        return type
return None
```

4.4.2 Identifying properties

The next phase is identifying the properties from the new source (the yellow part on Figure 4.2). After having identified the class of the new data source, we need to do the same with properties. In the case of the class, we used the name of the file, whereas here, we will use the columns: which columns contain properties and which ones contain class instances. For that, we use `maincol()` function. Given the class name and the columns, `maincol()` identifies which column contains the class instances. Logically, the rest of the columns would contain information about those instances meaning they would contain the class properties. The algorithm of the `maincol()` function is presented 3. Having identified the property columns, we need to check if such properties are already present in the ontology for that we also use the `closest()` function (algorithms 1, 2). If there is no existing property similar to the new property, we check if there is a property connecting the same classes as the new one; if yes, we assign to the new property URI of the found property, if not, we create a new URI.

Algorithm 3 maincol() function

```
Input: classname, column-names
columns ← column-names
for col in columns do
    score ← levenshtein-ratio(classname, col)
    if score > threshold then
        return col
```

4.4.3 Identifying property range

Now, we have identified the class and the properties of the new source. However, not all of the property columns will have an existing counterpart in the ontology, and for those which don't, we need to figure out where their range is another class or just a literal. As we know, every property has its domain and range, domain being the type of the subject and range being the type of the object. So, the domain is obviously the class which we have identified but the range is still unknown: it may be a class or a literal. For that, we compute the ratio of unique values in the property columns, if it is less than some threshold, in our case, we've set the threshold to 0.1, then the range of the property is a new class which we add to the ontology, if it is less than the threshold, the range is literal (`rdfs:Literal`).

4.4.4 Building the knowledge base

In the sections above, we discussed the logic of our methodology of creating a system for the ontology evolution. Sections 4.4.1-4.4.3 cover the main parts of the approach and key functions of the system. In this section, we discuss the very process of generating the ontology, building the knowledge base both for initial base model and evolution. Also, we justify the tools selected for the creation of the knowledge base.

For building the knowledge we used a pure Python library called RDFLib [rdf 2020]. The library has parsers and serialisers, and also it supports different formats RDF/XML, Turtle, N-Triples and others. The main advantage of RDFLib is that it is a library and not a tool. In our previous approaches, we were using tools like Protege,[Team 2015], Ontop[et al. 2020], [gra 2015], and neither of them had Python APIs that would allows us to automate the process of building the knowledge base. Whereas RDFLib provides us with flexibility in choices we make when creating a knowledge base. We used RDFLib for generating the schema (TBox) and loading the instances (ABox). Below, you can see a snippet from the TBox generating code using RDFLib.

```
# Generating TBox
self.graph = Graph()
self.gkb = Namespace('http://www.semanticweb.org/gkb#')
# Classes
self.graph.add((self.gkb.Unit, RDF.type, OWL.Class))
self.graph.add((self.gkb.Wellbore, RDF.type, OWL.Class))

# Unit properties
self.graph.add((self.gkb.located_in, RDF.type, RDF.Property))
self.graph.add((self.gkb.located_in, RDFS.domain, self.gkb.Unit))
self.graph.add((self.gkb.located_in, RDFS.range, RDFS.Literal))
```

So, as you can see, the process is quite intuitive. We create an instance of the `Graph` class, apply the `add()` method and pass the triples. In the above code, we are generating classes `Unit`, `Wellbore` and the properties of `Unit`. The same `add()` method is used for generating the ABox meaning loading the instances. The snippet from the ABox generation part is given below.

```
for _, row in formations.iterrows():
    formation_uri = self.gkb.Formation + '/' + neatstr(row['unit'])
    self.graph.add((formation_uri, RDF.type, self.gkb.Formation))
    self.graph.add((formation_uri, RDFS.label,
    Literal(row['unit'].strip())))
    self.graph.add((formation_uri, self.gkb.lithology,
    Literal(row['lithology'])))
    self.graph.add((formation_uri, self.gkb.age, Literal(row['age'])))
```

4.5 Query parsing

To enable the end user to retrieve answers from the knowledge base, we created a small in-own query parse using the Natural Language Toolkit(NLTK) library as base. The central idea was to help in the automation of the whole system by reducing the workload of the user to understand and write efficient SPARQL queries to retrieve the desired knowledge. The query parser acts an abstraction layer for the end-user since they are oblivious how the ontology looks in the back-end.

To throw some light on the internal working, we try to simplify the working through a flowchart.

The process takes in as input a natural language query(NLQ) from the user. For example, Where is Ekofisk formation?

- The input is tokenized to extract each word and thereby making the process of eliminating stop-words easier.
- We envision all questions related to stratigraphy to have formation as a subject. Hence, we proceed to locate the formation in the query. Once located, we check for its existence in our knowledge base. If not found, we record it in our log and notify system admins.
- If contained, we proceed further to retrieve all corresponding properties relating to the mentioned formation like location, age, lithology, etc.
- Next, we make an attempt to identify other classes like wellbores, fields, etc in the query. For example, Which well crosses the Ekofisk formation?
- If any other subject's presence is detected, we again check the instance's existence inside the knowledge base.
- We proceed to execute the query in case of successful search, else, we follow the same logging and notification process.

4.6 Ideas for future development

During the implementation of the current project, we had some ideas which could be implemented and bring some improvements to the current approach.

- **Class recognition improvement**. When the system processes a new file, it recognises some of the columns as properties or classes. That can be stored in a format of a dictionary where the key would be the recognised concept, and the value would be the list of names that were recognised as the concept. That data later can be used as training data for a more sophisticated system.For example:

  ```
  {Wellbore: [wlb, well, wlbdatda, wellbore_data],
  Formation: [...]}
  ```

- **Named-Entity-Recognition**. Use NER to improve the question parsing and identifying concepts from textual data.
- **R2RML mappings**. This is not necessary but as a side-feature could be implemented. For that, we can write a mapper function.

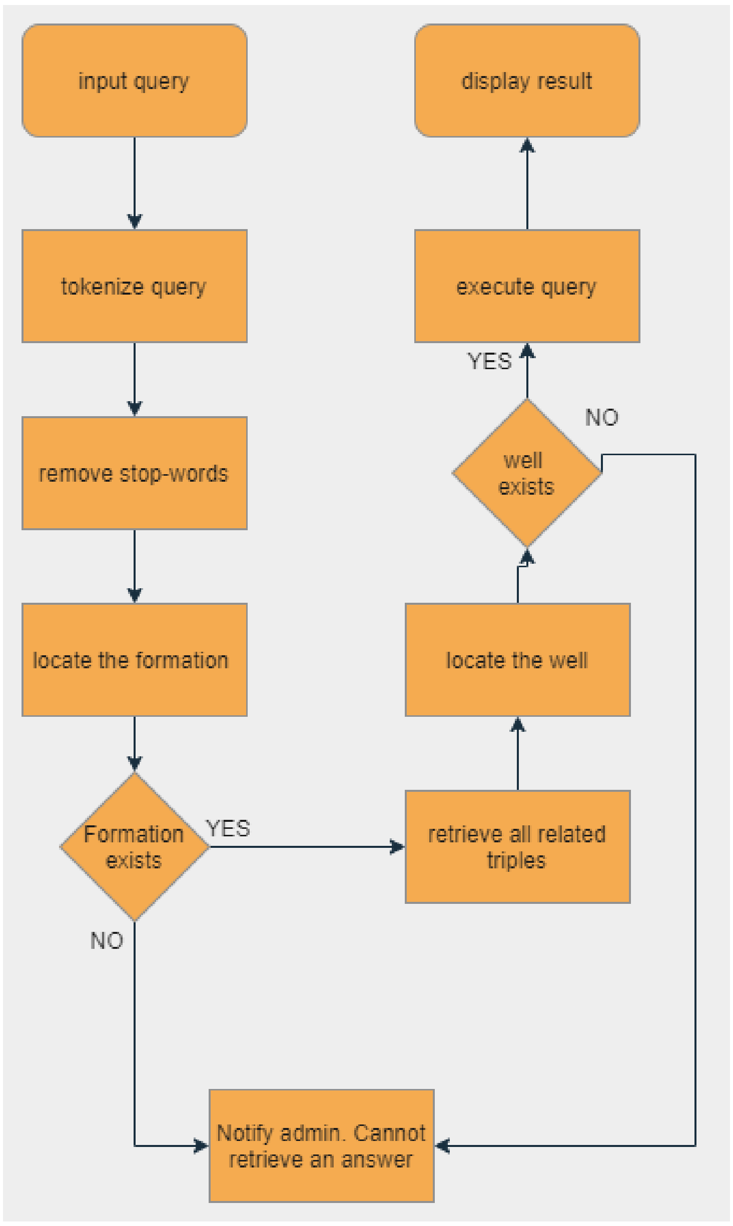


Figure 4.3: Caption

Chapter 5

Testing the evolution system

5.1 Setup

Both in the initial model generation and the evolution, we read 100 rows of each file. The files have not been cleaned, they may contain empty values. That was done to remove the manual step from the approach, and to see if the system is capable of running on almost raw files.

5.2 Initial model

Finally, let us present our approach in action. In this section, we will discuss the test cases of loading a feeding the new source to the system, and also we will discuss the results of the tests.
First, we need to generate our initial ontology which we have modelled in 4.3. The process of generation has been discussed in 4.4.4. On Figure 5.1, we can see the initial knowledge graph. For visualisation, we used one of the implementations of VOWL [Lohmann 2016] called WebVOWL[1]

[1]http://vowl.visualdataweb.org/

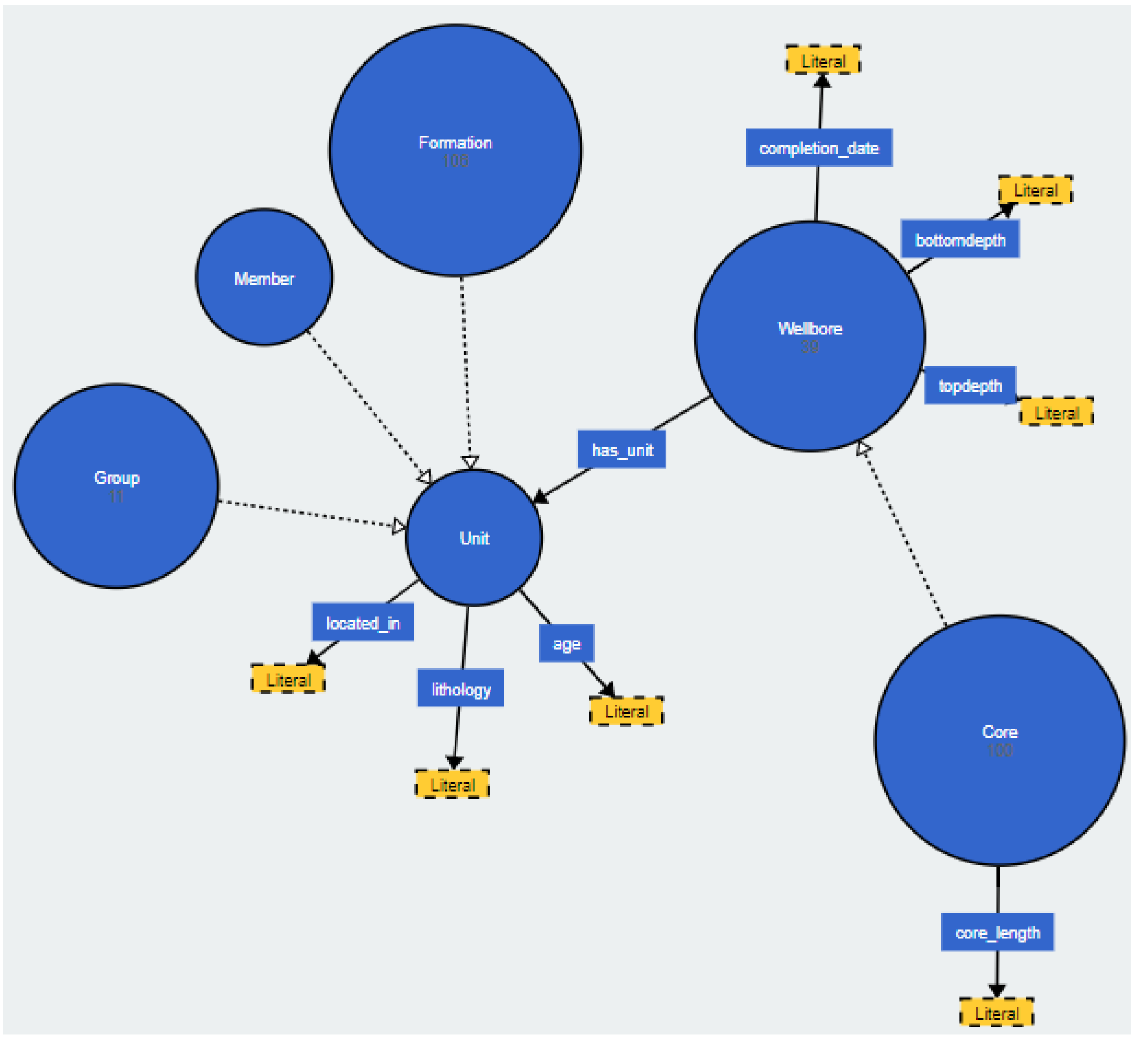


Figure 5.1: Initial knowledge base

5.3 Fields

Now, we will load a new file `field.csv` with column names:

```
field_name, operator, status, discovery_wellbore,
completion_date, situated_in, supply_base
```

and see if the knowledge base changes. On Figure 5.2, we can see that indeed the knowledge base changed, the system has identified new classes and added them to the ontology.

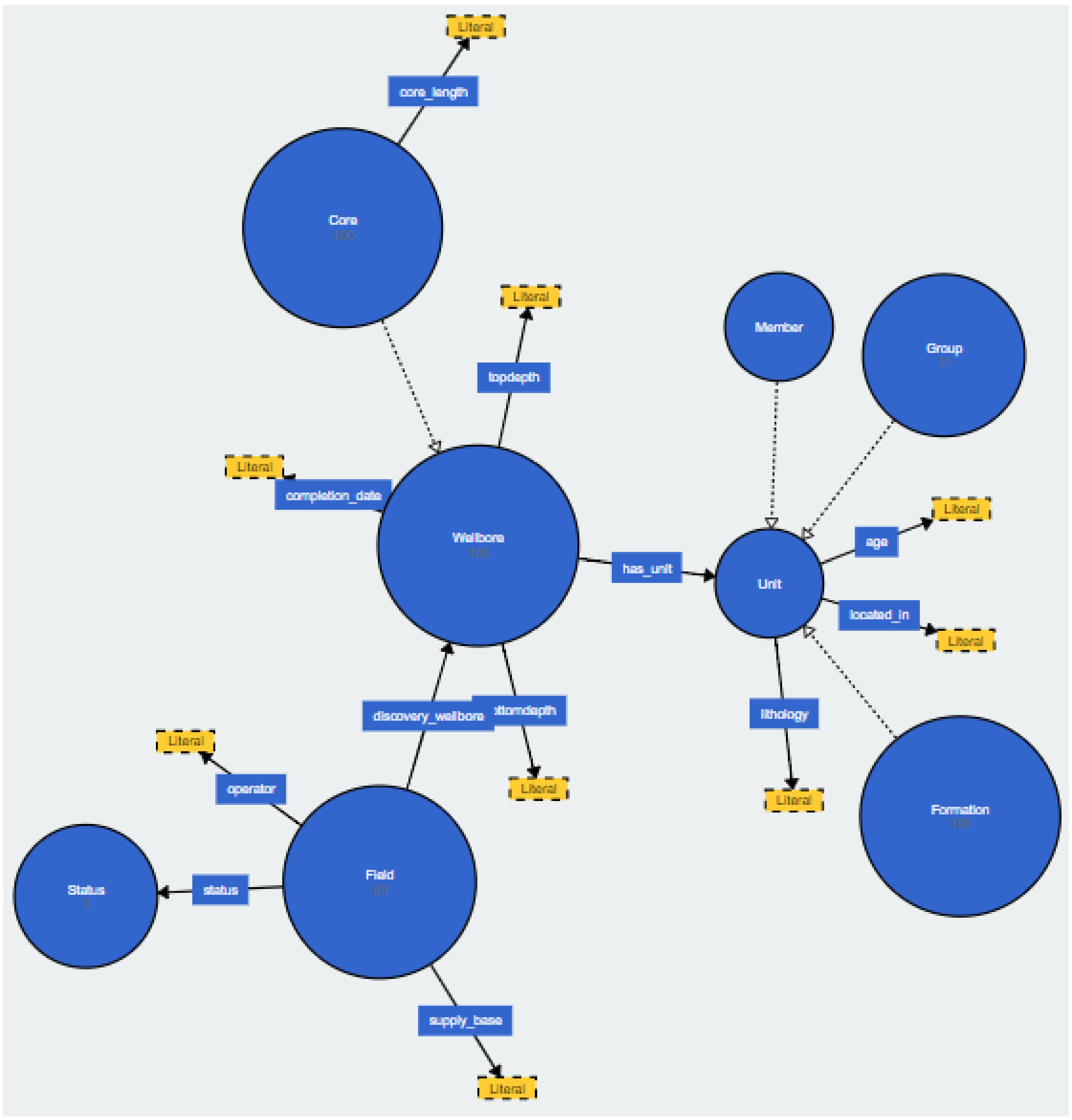


Figure 5.2: The knowledge base after processing a new file field.csv

To see the changes in more detail, refer to the table 5.1. It contains the statistics of the knowledge base before and after feeding the file.

	Before	After
Classes	6	8
Object properties	1	3
Data properties	7	9
Individuals	256	397
Nodes	13	17
Edges	12	16

Table 5.1: Knowledge base statistics before and after evolution

As we can, see, 2 new classes (`Field`, `Status`) have added, 2 new object property(`status`, `discovery_wellbore`), 2 new data properties(`operator`, `supply_base`) 141 new instances, 4 more nodes and 16 more edges. The process of evolution created However, it would be also interesting to see if some of the existing classes and properties were identified and complemented with new individuals. Indeed, class `Wellbore` was identified as the range type of `discovery_wellbore` property and was supplemented with 69 new individuals.

The identified concepts and properties:

- column `Completion_date` was recognised as `completion_date` property which is present in the ontology;
- column `discovery_wellbore` was identified as a new property but it's range was recognised as class `Wellbore`;
- column `situated_in` was recognised as property `located_in`.

5.4 Discoveries

We will continue with our experiment and pass another file to the ontology and see what changes the new source will bring. This time, we are loading a file `discovery.csv`. The file has the following columns:

```
discovery_name, operator, status,
resource, wellbore, site, field, year.
```

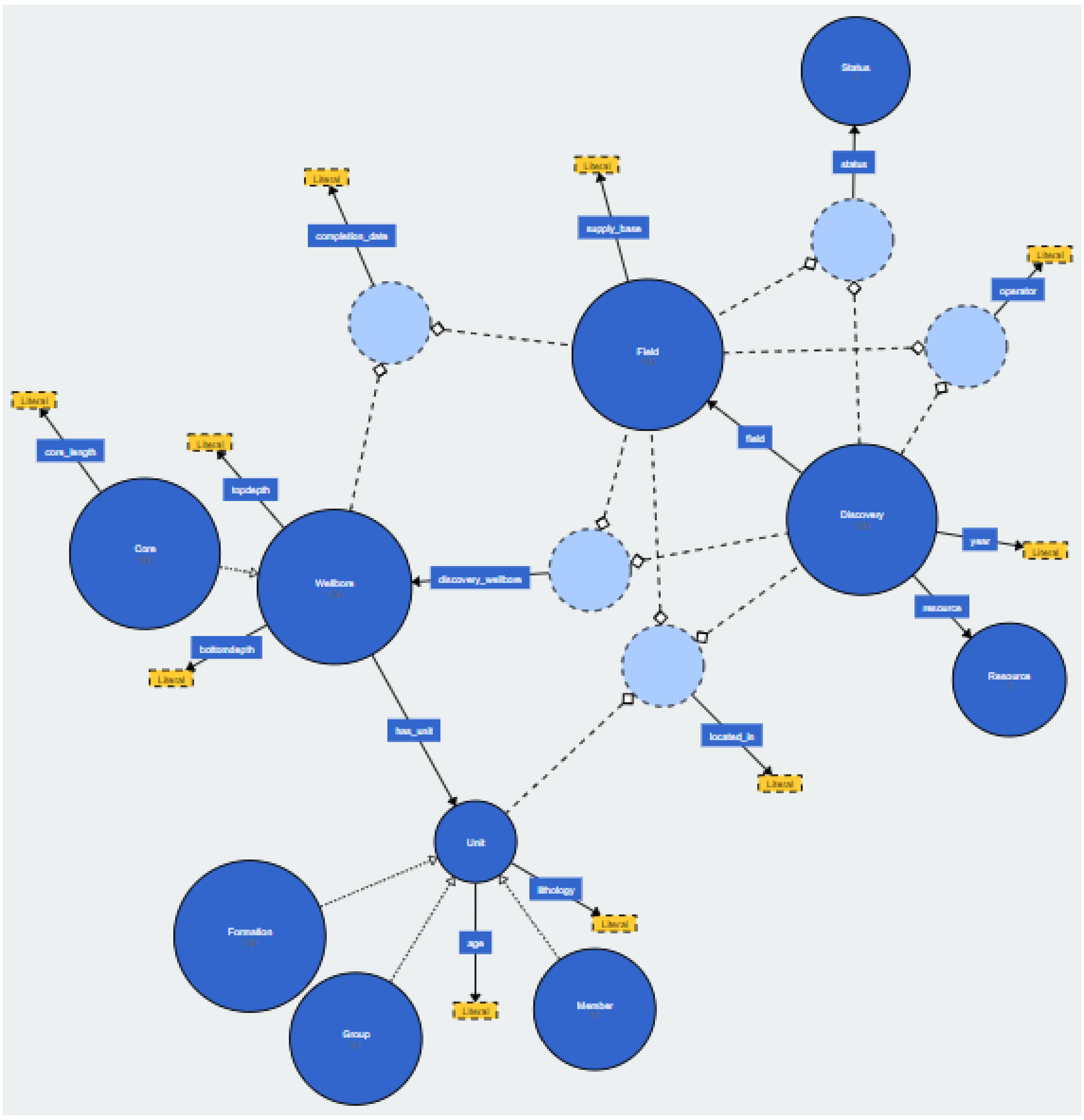


Figure 5.3: The knowledge base after processing a new file discovery.csv

After passing the file to the system, the ontology evolved, and new classes and properties were created (Figure 5.3). More detailed statistics is provided on the table 5.2

	Before	After
Classes	8	15
Object properties	3	5
Data properties	9	10
Individuals	397	571
Nodes	17	25
Edges	16	30

Table 5.2: Knowledge base statistics before and after evolution

Identified concepts and properties:

- column `operator` was identified as property `operator` which is present in the class;
- column `status` was identified as property `status` which is present in the class;
- column `wellbore` was identified as property `discovery_wellbore` which is present in the class;
- column `site` was identified as property `located_in` which is present in the class;
- column `operator` was identified as property `operator` which is present in the class;

5.5 Complementary files

After loading two new files into the ontology, several new classes and properties were generated. Now, we will pass the files which do not represent any new class but contain additional information about the existing classes. The incoming files are:

```
wellbore_exploration_all.csv:
wellbore_name,operator,status,resource_content,start_date,
completion_date,field,discovery,age,formation,located

field_description.csv:
field_name, description

discovery_description.csv:
discovery_name,description
```

The final ontology is shown on Figure 5.4

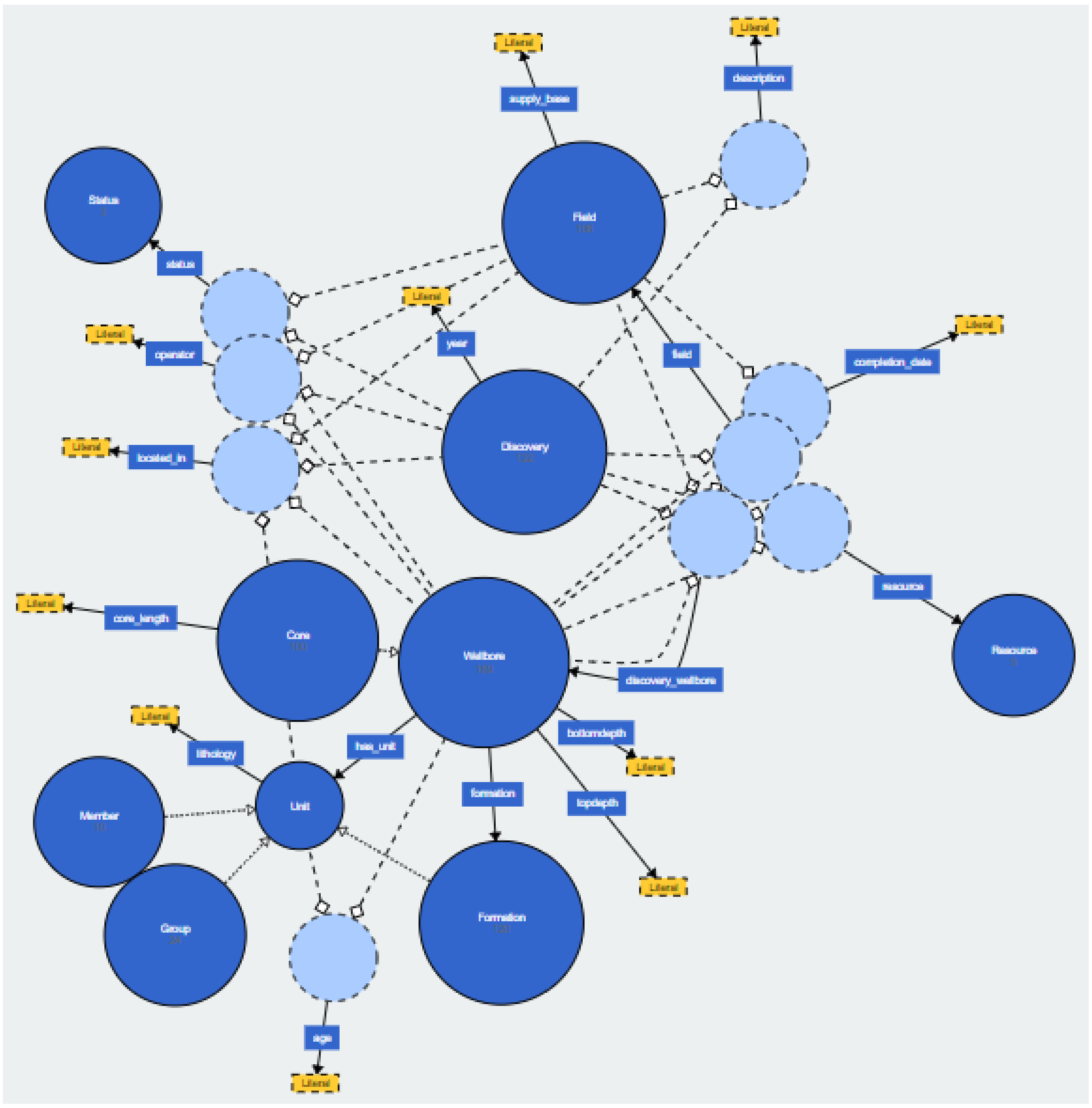


Figure 5.4: Final ontology after evolution

Identified concepts and properties:

- file `wellbore_exploration_all` was identified as class `Wellbore` which is present in the class;
- column `operator` was identified as property `operator` which is present in the class;
- column `status` was identified as property `status` which is present in the class;

- column `resource_content` was identified as property `resource` which is present in the class;
- column `field` was identified as property `field` which is present in the class;
- column `discovery` was identified as property `discovery_wellbore` which is present in the class;
- column `located` was identified as property `located_in` which is present in the class;
- file `field_description` was identified as class `Field` which is present in the class;
- file `discovery_description` was identified as class `Discovery` which is present in the class.

On Table 5.3, we can see the exact numbers of the final evolution step.

	Before	After
Classes	15	19
Object properties	5	6
Data properties	10	11
Individuals	571	657
Nodes	25	30
Edges	40	44

Table 5.3: Knowledge base statistics before and after evolution

5.6 Overall Results

After having passed all of the files to the system, ontology has gone through some changes. To summarise all of the points discussed in the above sections, we provide tables 5.4, 5.5.

	Recognised	Created
Classes	2	13
Properties	8	9
Property ranges	2	5

Table 5.4: Classes and properties recognised and created during the tested evolution

	Classes	Object properties	Data properties	Individuals	Nodes	Edges
base model	6	1	7	256	13	12
after field	8	3	9	397	17	16
after discovery	15	5	10	571	25	30
after complementary files	19	6	11	657	30	44

Table 5.5: Knowledge base statistics comparison for each step of evolution

5.7 Query parsing tests

We demonstrate a few questions which we used as cornerstone while building and testing the query parser. For the MVP, we keep the range and nature of questions limited to close-ended ones. Leaving room for open ended questions in the future as per the approval and choice of the stakeholders. The prototype in the figure 5.5 is a simple yet effective version of our idea, with plans to improve it in the pipeline.

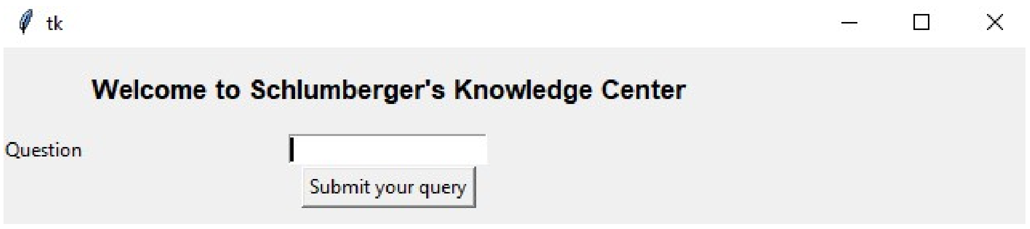
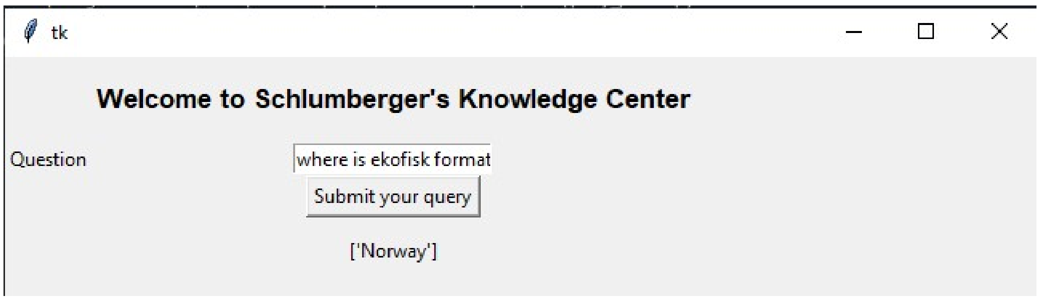


Figure 5.5: Prototype of the query parser

The examples with the parser was tested with are as follows, containing the query inside the parser along with the displayed answer:

- Where is the Ekofisk formation(fig 5.6)?

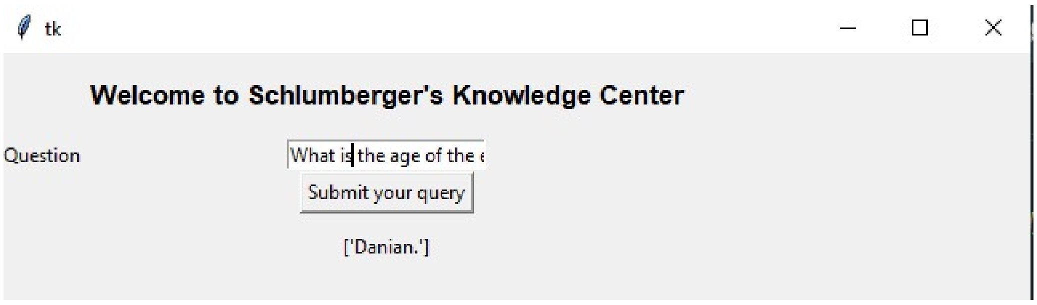


where is ekofisk format

['Norway']

Figure 5.6: Location of a formation

- What is the age of Ekofisk formation(fig 5.7)?


tk

Welcome to Schlumberger's Knowledge Center

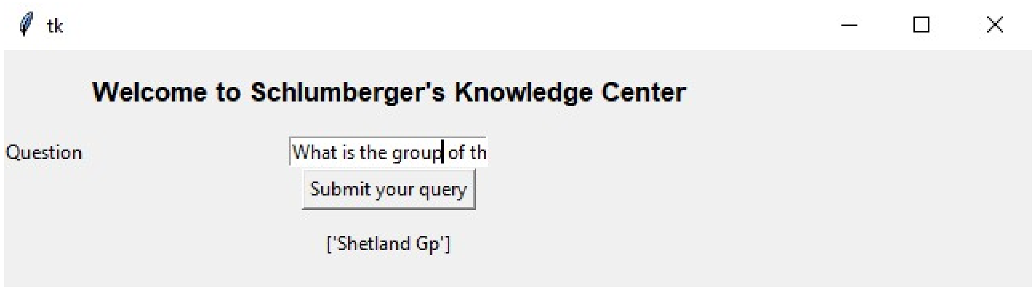
Question What is the age of the e

Submit your query

['Danian.']

Figure 5.7: Age of a formation

- Which group does the Ekofisk formation belong to(fig 5.8)?


tk

Welcome to Schlumberger's Knowledge Center

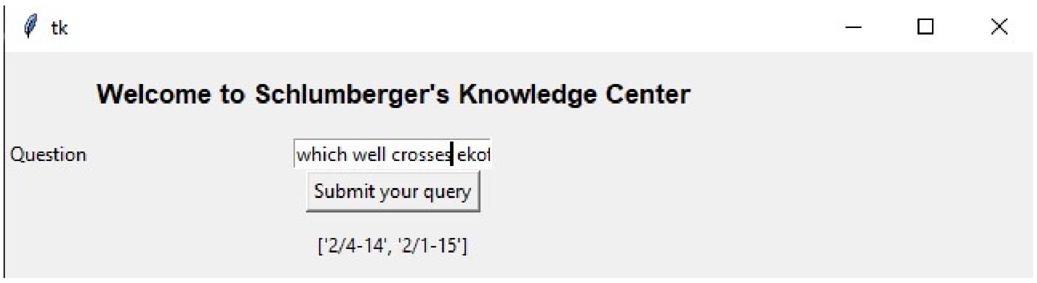
Question What is the group of th

Submit your query

['Shetland Gp']

Figure 5.8: Parent Group of a formation

- Which well crosses the Ekofisk formation?(fig 5.9)?


tk

Welcome to Schlumberger's Knowledge Center

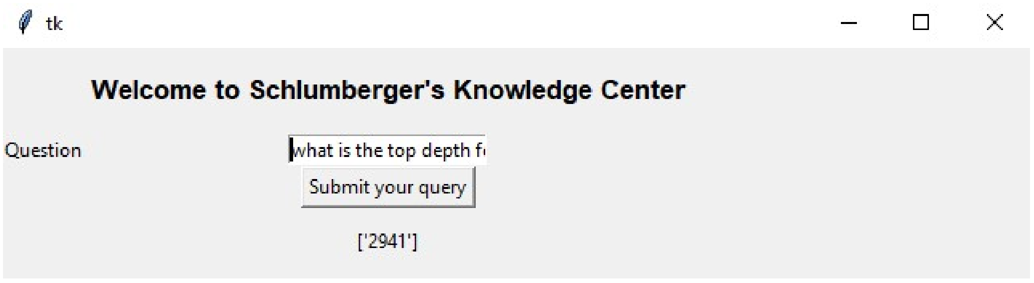
Question which well crosses ekof

Submit your query

['2/4-14', '2/1-15']

Figure 5.9: Wells belonging to a formation

- What the top depth of the well 2/1-15 located in the Ekofisk formation(fig 5.10)?


tk

Welcome to Schlumberger's Knowledge Center

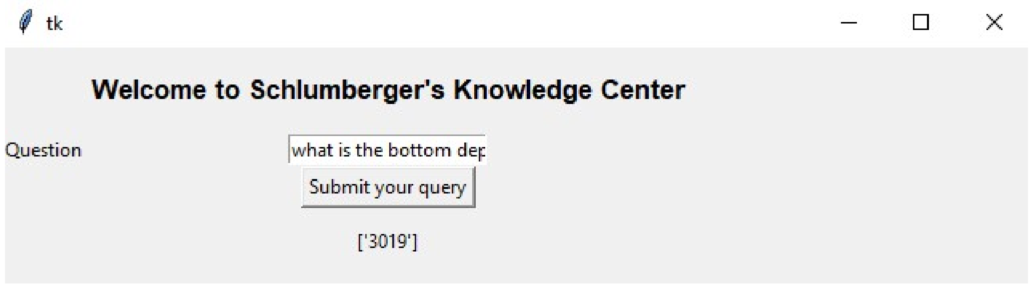
Question what is the top depth f

Submit your query

['2941']

Figure 5.10: Top-depth of well contained in a formation

- What the bottom depth of the well 2/1-15 located in the Ekofisk formation(fig 5.11)?


tk

Welcome to Schlumberger's Knowledge Center

Question what is the bottom dep

Submit your query

['3019']

Figure 5.11: Top-depth of well contained in a formation

The major advantage of having the query parser is the reduced time to formulate and execute a SPARQL query, The end-user need not neither require any require information about the SPARQL language nor any idea on the current ontology. Yes, certain constraints would be required to be adhered to achieve expected results but it is small trade-off to learning and executing SPARQL queries.

Chapter 6
Conclusion

In this project, we addressed the problem of creating an ontology which would evolve over time. After two unsuccessful attempts, we were able to develop an approach which if not solves the problem, then gets the closest to the solution. The solution we propose in this project is tool-free meaning it is purely programmatic. That gives the approach flexibility and that what allowed us to make the process of ontology evolution automatic. While searching for solution, trying and testing different ideas, approaches and tools, we have learnt about the domain, the theory, about the state-of-the-art developments, different tools and libraries. Apart from technical knowledge, we have gained some experience in the process of doing researches, we learnt about the positive and negatives sides.
Our proposed solution is far from perfect but we believe it will help to achieve the final goal of the stakeholders.

Bibliography

[Cullot 2007] Nadine Cullot, Raji Ghawi and Kokou Yetongnon. DB2OWL: A tool for automatic database-to-ontology mapping. pages 491–494, 01 2007. (Cited on page 5.)

[et al. 2020] Xiao G. et al. The Virtual Knowledge Graph System Ontop. Lecture Notes in Computer Science, vol. vol 12507, 2020. (Cited on pages 8, 9 and 18.)

[gra 2015] GraphDB. https://graphdb.ontotext.com/documentation/free/about-graphdb.html/, 2015. Accessed:2021-19-01. (Cited on pages 7 and 18.)

[Huang 2019] Chengxi Huang, Hongming Cai, Lida Xu, Boyi Xu, Yizhi Gu and Lihong Jiang. Data-driven ontology generation and evolution towards intelligent service in manufacturing systems. Future Generation Computer Systems, vol. 101, pages 197–207, 2019. (Cited on page 6.)

[Lohmann 2016] Steffen Lohmann, Stefan Negru, Florian Haag and Thomas Ertl. Visualizing Ontologies with VOWL. Semantic Web, vol. 7, no. 4, pages 399–419, 2016. (Cited on page 21.)

[Milos Chaloupka 2020] Martin Necasky Milos Chaloupka. Using Berlin SPARQL Benchmark to Evaluate Relational Database Virtual SPARQL Endpoints. SWJ, 2020. (Cited on page 8.)

[rdf 2020] RDFlib. https://rdflib.dev/, 2020. Accessed: 2021-19-01. (Cited on page 18.)

[Team 2015] Musen MA; Protégé Team. The Protégé Project: A Look Back and a Look Forward. AI Matters, vol. 1, no. 4, pages 4–12, 2015. (Cited on pages 9 and 18.)

[Tijerino 2005] Yuri Tijerino, David Embley, Deryle Lonsdale, Yihong Ding and George Nagy. Towards Ontology Generation from Tables. World Wide Web, vol. 8, pages 261–285, 09 2005. (Cited on page 5.)