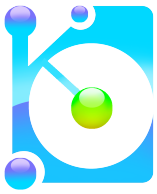

WP3-D1

Generated Key Queries for the Case Studies

Project full title:	Knowledge Driven Data Exploitation
Project acronym:	K-Drive
Grant agreement no.:	286348
Project instrument:	EU FP7/Maria-Curie IAPP/PEOPLE WP 2011
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IBM, UNIABDN/WP3-D1/D/PU/b3
Responsible editors:	Jing Mei, Yuan Ren
Reviewers:	[...]
Contributing participants:	IBM, UNIABDN
Contributing workpackages:	WP3
Contractual date of deliverable:	31 March 2013
Actual submission date:	31 March 2013

Abstract
The goal of this deliverable is to investigate and identify the key queries in the two case studies. To achieve this, we first present a generic framework to describe the diffferent aspects of datasets and queries. Then we apply such a framework on the two case studies, incorporating the characteristics of datasets and opinions of domain experts. At the end, we manually generate a set of key queries for these two case studies.

Keyword List
query generation, SPARQL querying

Project funded by the European Commission within the 7th Framework Programme/Maria Curie Industry-Academia Partnerships and Pathways schema/PEOPLE Work Programme 2011.

Generated Key Queries for the Case Studies

Yuan Ren[1] **and Jing Mei**[2]

[1] Department of Computing Science, Aberdeen University, UK
Email: y.ren@abdn.ac.uk

[2] IBM Research-China, Beijing, China
Email: meijing@cn.ibm.com

31 March 2013

Abstract
The goal of this deliverable is to investigate and identify the key queries in the two case studies. To achieve this, we first present a generic framework to describe the diffferent aspects of datasets and queries. Then we apply such a framework on the two case studies, incorporating the characteristics of datasets and opinions of domain experts. At the end, we manually generate a set of key queries for these two case studies.

Keyword List
query generation, SPARQL querying

Contents

1 Introduction

Semantic query answering with SPARQL query language against RDF datasets is an important means to exploit semantic datasets such as linked data or biomedical ontologies. There has been an abundance of research on answering semantic queries, but only limited attention has been paid to construct queries that can most effectively retrieve the insightful information in a dataset.

Indeed, query generation is a non-trivial problem on the Semantic Web and in K-Drive. From the users' perspective, they are not always familiar with Semantic Web technologies such as RDF/SPARQL. They are also not always familiar with the datasets they are dealing with, especially when the datasets are of large scale and linked by/to many remote datasets. From a knowledge representation's perspective, queries are important means for describing the contents of the datasets as their solutions correspond to subsets of the datasets that satisfy certain constraints. In this regard, an insightful query usually reveals some meta-knowledge, such as topics, trends, of semantic datasets.

Query generation (QG) has been studied in the field of database with the main motivation of testing databases. Some QG approaches (such as [**?**]) are based on database schemas, while others (such as [**?**]) are based on actual data in databases. A related research problem is query recommendation (QR), where query logs are widely used to generate queries based on querying and browsing behaviours of users. These approaches (such as [**?**]) rely on the knowledge about users. Similar approaches (such as [**?**]) are also used in information retrieval.

There have been some work on QG in the field of Semantic Web. Similar to the work in database, most of the existing work is for testing semantic web engines. For instance, Cuenca Grau and Stoilos [**?**] presented an approach to generating queries to test incomplete ontology reasoners for EL, DL-Lite and DLP. Their generated queries are *based on ontologies rather than actual data*. Görlitz et al. [**?**] proposed to generate queries for testing linked data query engines, based on some *input parameters*.

Nevertheless, both of these approaches are designed to generate queries for evaluating the quality or performance of implementations instead of assisting users in exploitation of semantic data. As far as we know, there is no existing work on systematic generation of semantic queries, based on analysis of given semantic data, for the purpose of facilitating users to understand the data with insightful queries.

One of the major difference between the above existing work and our work presented in this deliverable is that instead of focus on the evaluation of the system, we are more interested in the revelation of interesting information in the datasets to users. As a consequence, our approach will focus on the characteristics and insights of the datasets, as well as people's requirements to queries.

In the context of K-Drive, query generation facilitates exploitation of large sale data, helping users to understand the contents of datasets, and to identify the most relevant ones. The work presented in this deliverable will be the foundation for future research on query generation and data exploitation. Particularly, WP5 *Stream Query Generation* will tackle automatic query generation dynamically. WP6 Stream Semantic Querying will answer such queries against dynamic data. WP8 Hypothesis Generation and WP9 Guidance will extend the query generation technologies to produce hypothesis and guidance information for users.

The deliverable is organised as follows: in Sec. **??** we will introduce the basics of RDF and SPARQL. In Sec. **??** we propose a query generation framework that can be used to analyse datasets and parameterise queries. Section **??** applies this framework on datasets used in case studies and generates several typical queries. The last section concludes the deliverable and indicate potential next steps.

2 Background

In this section, we briefly introduce the notion of RDF documents and SPARQL queries to facilitate the understanding of the remaining of the deliverable. For the sake of readability and conciseness, we present the general ideas about RDF and SPARQL and use examples to demonstrate their usage. More formal and comprehensive introductions about the technical details of RDF and SPARQL can be found in their corresponding documentations. [1][2].

2.1 RDF

Resource Description Framework (RDF) [**?**] is one of the most widely used data interchange formats on the Semantic Web. An RDF document uses a graph data model. In such a graph, there are three mutually disjoint types of nodes:

1. An **IRI** (Internationalised Resource Identifier) is a sequence of characters from the Universal Character Set (Unicode/ISO10646). It uniquely identifies a resource on the web with a url-like string. For example, the IRI $http://www.kdrive\text{-}project.eu/personal/jeff\_z\_pan$ can be used as an identifier of Dr. Jeff Z. Pan (as a resource on the web). A resource can have multiple IRIs.

2. A **literal** is used to identify values such as numbers and dates by means of a lexical representation. It serves as a more intuitive and convenient alternative of IRI for such values. For example, "University of Aberdeen", "ISOCO@en", "TRUE", "532" are all valid literals.

3. A **blank node** is a resource not given an IRI reference.

Given a set of IRI reference $\mathcal{R}$, a set of literals $\mathcal{L}$, a set of blank nodes $\mathcal{B}$, an RDF statement is a triple $\langle s, p, o\rangle$ on $(\mathcal{R} \cup \mathcal{B}) \times \mathcal{R} \times (\mathcal{R} \cup \mathcal{L} \cup \mathcal{B})$, where s, p, o are the subject, predicate and object of the triple, respectively. An RDF document is a set of triples.

There are different syntaxes for RDF documents, such as RDF/XML, Turtle, N3. The following is an example of an RDF document in RDF/XML syntax [3]:

Example 1 ***(RDF Document)***

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:uoa="http://www.abdn.ac.uk/"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:owl="http://www.w3.org/2002/07/owl">
  <rdf:Description rdf:about="http://www.abdn.ac.uk/JPan">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <foaf:name>Jeff Z. Pan</foaf:name>
    <foaf:title>Dr</foaf:title>
    <foaf:homepage
rdf:resource="http://homepages.abdn.ac.uk/jeff.z.pan/pages/"/>
```

[1] http://www.w3.org/TR/rdf-primer/

[2] http://www.w3.org/TR/rdf-sparql-query/

[3] The example is adapted from the FOAF DescribingAPerson example http://wiki.foaf-project.org/w/DescribingAPerson

```
    <foaf:schoolHomepage rdf:resource="http://abdn.ac.uk/"/>
    <owl:sameAs
rdf:resource="http://www.kdrive-project.eu/personal/jeff_z_Pan"/>
  </rdf:Description>
</rdf:RDF>
```

This is a rdf document describing the Dr. Jeff Z. Pan. It tells the information such as name, title, homepage, about the person and refers to a different IRI of the same person.

As we can see from the example, IRIs can be defined in different namespaces. In Example **??**, we have namespaces such as rdf, $foaf$ and owl. An important advantage of such design is that information described in remote RDF documents can be used to enrich local data. And widely used common namespaces become a natural foundation of data integration. Different namespaces can also be connected via labelled links. For example, in the above document we have triple $\langle uoa : JPan, rdf : type, foaf : Person\rangle$. The subject, predicate and object of this triple come from three different namespaces. RDF has its own namespace as shown in the example. In this namespace, it introduces predicates such as `rdf:type`, indicating that the subject of the triple is an instance of the object of the triple.

Due to the graph data model, an RDF document can also be equivalently represented as a graph. Particularly, given an RDF document D, there is a unique directed, labeled graph $G = \langle N, E\rangle$, where N is the set of all nodes in D, and $E \subseteq N \times N$ contains an edge e labelled p from one node s to another node o *iff* there is $\langle s, p, o\rangle \in D$.

2.2 SPARQL

The W3C recommendation SPARQL (SPARQL Protocol AND RDF Query Language) is the de facto standard query language for RDF-based ontologies. It serves a role to the Semantic Web similar as the role SQL serves to relational databases. Below is an example of a SPARQL query:

Example 2 ***(SPARQL query)***

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdf:<http://xmlns.com/foaf/0.1/>
SELECT ?person
WHERE {
   ?person foaf:schoolHomepage <http://abdn.ac.uk/> .}
```

The above very basic example retrieves all persons whose school's homepage is `http://abdn.ac.uk`. SPARQL queries are executed by matching the triple patterns in the query to the triples in a dataset. To answer the query, a query engine will substitute variable `?person` by a subject in the dataset and check if the substituted triple exists in the dataset. Note that the dataset does not have to be the original RDF document. It can also be an entailment closure of the RDF document w.r.t. a particular entailment regime[4]. For example, if we use the *simple entailment regime* to query against the RDF document in Example **??**, then the pattern will be matched against the original RDF graph. Hence `http://www.abdn.ac.uk/JPan` will be the only answer. If we use the *OWL 2 Direct Semantics Entailment Regime*, then the pattern will be matched against the OWL materialisation closure of the RDF graph. Hence in addition to the previous answer, `http://www.kdrive-project.eu/personal/jeff_z_Pan`

[4] `http://www.w3.org/TR/2013/REC-sparql11-entailment-20130321/`

will be another answer. To answer a SPARQL query, an inference engine of the corresponding entailment regime is needed.

SPARQL supports many different patterns. The query in Example **??** uses the basic graph pattern (BGP), whose query body consists solely of a set of triples with variables. In addition to BGP, there are many other filters and modifiers.

Some of the most widely used ones are listed below, for more comprehensive definition and examples of SPARQL filters and modifiers, we refer to the SPARQL recommendation documentation: [5]

- **String Value Restriction:** For example, `FILTER regex(?variable exp)` can be used to exclude answers in which the string literal value of `?variable` does not match the regular expression `exp`;
- **Numeric Value Restriction:** For example, `FILTER (?variable < number)` can be used to exclude answers in which the numeric literal value of `?variabel` is not smaller than the `number`;
- **Non-existence:** For example, given a BGP `bgp`, the `FILTER NOT EXISTS` $\{$`bgp`$\}$ pattern can be used to exclude answers that satisfy `bgp`;
- **Optional Patterns:** For example, given a BGP `bgp`, the `OPTIONAL` $\{$`bgp`$\}$ pattern answers the query by treating `bgp` as a optional constrain;
- **Alternative Patterns:** For example, given two BGPs `bgp1` and `bgp2`, the $\{$`bgp1`$\}$ `UNION` $\{$`bgp2`$\}$ pattern answers the query with the union of answers to `bgp1` and `bgp2`;

There are four forms of SPARQL queries:

1. **SELECT:** Returns all, or a subset of, the variables bound in a query pattern match. The example we showed is a SELECT query. Such query is also usually called a conjunctive query. The answer to a SELECT query can be represented by a variable-value pair set $\{(?variable, value)\}$ with each variable paired with the resource or literal it is bound to;
2. **ASK:** Returns a boolean indicating whether a query pattern matches or not. This is also usually called a boolean query. The answer to a ASK query is either true or false;
3. **CONSTRUCT:** Returns an RDF graph constructed by substituting variables in a set of triple templates;
4. **DESCRIBE:** Returns an RDF graph that describes the resources found;

In later sections, we will revisit these features of SPARQL when investigating the key queries of use cases.

3 Query Generation Framework

In the context of the K-Drive project, we define *query generation as a procedure of identifying a set of queries, whose relations and solutions provide insights of the datasets to users*. From this high-level definition we can see that query generation has the following concerns:

[5] `http://www.w3.org/TR/sparql11-query/`

1. **Insightfulness:** the queries generated should reveal certain characteristics of the dataset, either syntactic, or semantic. Hence the query generation should not be compromised by the difficulty of generating and answering such queries. Note that, query insightfulness can be delivered by not only individual quereis, but also the relations between multiple queries, particularly when the solution of certain queries have strong correspondences. Such insightfulness will be discussed in Sec. **??** of this deliverable;

2. **Comprehensibility:** the queries generated and their relations and solutions must be comprehensible to users, otherwise the insights can not be successfully grasped by users, making query generation pointless. Such comprehensibility can be achieved, on the one hand, use of intelligent interfaces, and on the other hand, control of the syntactic forms of generated queries. Indeed, if the generated queries are too lengthly or too complicated, it will be very difficult for people to understand, even with well designed interface. The design and development of intelligent interfaces will be investigated in WP4 and is out of the scope of this deliverable. The control of the query parameters will be discussed in Sec. **??** of this deliverable;

3. **Query Answering:** the insights of the datasets are delivered to users by answering the generated queries. This implies that query answering is an important following-up task of query generation. This task is out of the scope of this deliverble and will be discussed in WP6.

3.1 Query Insightfulness

Before investigating the insightfulness of queries, we first examine the relations between queries, their solutions and datasets. Based on this analysis, we focus on a certain form of query. Then we discuss what kind of insights can be discovered with the generation of such queries.

3.1.1 Queries and Sub-graphs

A dataset, as we explained at the end of Sec. **??**, can be represented as a graph. Similarly, a basic graph patterns (BGP) in a SPARQL query can also be represented as a graph. Particularly, given a BGP who consists of a set of triples, a unique directed, labelled graph can be constructed in a similar way by treating variables as nodes as well. Let D be an RDF graph, q be a SELECT query of a BGP, $Ans(q)$ be the set of answers of q and G be the graph for the BGP. For each answer $a \in Ans(q)$, let G_a be the graph of substiting variables in G with resources or literals according to a, then it is obvious that G_a is a sub-graph of D, which means that the nodes and edges of G_a are both subsets of nodes and edges of D, respectively.

Such a observation can be extended to cover other filters and modifiers because they only restrict or extend the answer set $Ans(q)$. We can further extend such an observation to other query forms as follows:

1. **SELECT** query answers correspond to sub-graphs of the dataset, as we explained above;

2. **ASK** query answer is true iff there is a sub-graph of the dataset that matches the query pattern. This is apparent due to the answering mechanism of ASK query: an ASK query q_1 can be reduced to a SELECT query q_2 with the same query pattern such that $Ans(q_1) = true$ iff $Ans(q_2)! = \emptyset$. In this case, there must be a sub-graph of the dataset that correspondes to an answer in $Ans(q_2)$. Consequently the sub-graph matches the pattern of q_1.

3. **CONSTRUCT** query answer is an RDF graph constructed w.r.t. the query. Although this graph does not have to be a sub-graph of the queried dataset, its existence is determined by the existence of some sub-graph of the dataset that matches the query pattern. Nevertheless, the relation between the matched sub-graph and the constructed graph is characterised by the query but not the dataset itself.

4. **DESCRIBE** query answers correspond to sub-graphs of the dataset that are related to the resources found by the query. Such a description is usually determined by the maintainer of the dataset. This implies that there is a maintainer who already knows which insights should be extracted from the dataset.

From the above analysis, we realise that the insights of the dataset alone might not be sufficient to generate CONSTRUCT queries as knowledge outside of the dataset will be needed to translate the matched sub-graphs to constructed graphs. Also, it might be unnecessary to generate DESCRIBE queries since the retrieved insights, i.e. the descriptions, is already known to some maintainer of the dataset. Furthermore, the ASK queries are reducible to SELECT queries. Hence, in this deliverable we will focus on the generation of the SELECT queries. This is actually not surprise because SELECT queries are also widely used in relational databases and information retrieval systems, and are mostly familiar to our users. As we will see in later sections, our investigation with use case partners also suggest that their target user groups are mostly interested in SELECT queries.

3.1.2 Insights of Sub-graphs

Given that a SELECT query corresponds to a set of sub-graphs of the dataset in question, to generate queries is actually to find sub-graphs of the dataset that contains some insights. From the perspective of K-Drive, such insights emerge from the contrast in the following dimensions:

1. **Nominative v.s. topological information:** The sub-graphs, as well as the RDF graph they belong to, contain two major types of information. One is the nominative information about what kind of resources and literals are talked about in the graph, regardless how they interact with each other. The other is the topological information about how different nodes in the graph are connected to create a certain shape. The former is about the topics of the graph, while the later is about the structure of the graph. Generated queries should cover both kinds of information:

    - Dataset topics: **topic queries** that discover frequently specified entities in a dataset, which can be regarded as the central topics of a dataset. A topic query can also be regarded as a straightforward kind of DESCRIBE query as it describes the retrieved entities with related resources or literals;
    - Dataset structures: **structural queries** that discover frequently appearing structural patterns in a dataset. Most widely considered structural patterns include star-shape, tree-shape, circle-shape. Depending on the dataset and user requirement, such shapes have different levels of significance in structural query generation;

2. **Single v.s. multiple sub-graphs:** Each SELECT query corresponds to a set of sub-graphs of the dataset, then several SELECT queries correspond to multiple sets of sub-graphs of the dataset. It is natural to ask, is there any correspondence between these sets of sub-graphs? This leads to the generation of the following queries:

- High correspondence: if the matched sub-graphs of two queries are very similar, or one query's results are most subsumed by another query's results, then these two queries have high correspondence. This pair of **corresponding queries** reveal the correspondence between different properties of same entities;
- Low correspondence: if only few sub-graphs can be matched to both two queries, then these two queries have low correspondence. This pair of **exclusive queries** reveal that some entities may have drastically different properties. Of course, arbitrary queries are most likely exclusive to each other, e.g. a query for cars and a query for people. So in generation practice we are more interested in non-trivial queries that share certain properties but differ on others, e.g. a query for people that have low body weight and a query for people that have diabete;
- Exceptions: if a SELECT query of pattern q_1 has high correspondence to another SELECT query of pattern q_2, then it is obvious that sub-graphs that matche q_1 but not q_2 are exceptions compared to those more common sub-graphs that match both. Such **exception queries** can be generated as $\{q_1$, `FILTER NOT EXISTS`$\{q_2\}\}$. Similar when q_1 and q_2 make exclusive queries, an exception query $\{q_1, q_2\}$ can be generated;

3. **Static v.s. dynamic dataset:** As motivated by our use cases, the dataset of an application may change. As a consequence, the sub-graphs that correspond to the query may also change. Despite the evoluation of dataset, there can be also sources of dynamics. In this deliverable, we are mainly interested in the following:
   - Temporal dynamics: As said above, **temporally dyanmical queries** have drastically different answers at different points of time. Such difference releaves the pattern of data evolution in the dataset. These queries can be generated to contineously monitor the dataset;
   - Spatial dynamics: linked data comprises datasets from different sources. **Spatially dynamical queries** have drastically different answers on different sub-datasets. Such difference reveals the diversities among different data sources;
4. **Static v.s. dynamic queries:** It is obvious that when the query changes, the answers should also change (except that the query changes to a corresponding query). However, there are situations where the changes between the answers and the queries have an obvious pattern:
   - Value dynamics: **Value dynamical queries** have different answers strongly corresponded to the change of particular values in the queries. For example, the likelyhood of having diabete usually grows with the body weight of person. It is similar to the high/low correspondences of queries but in this case, the level of correspondence changes;
   - User dynamics: Query answers are computed against a dataset, which can be an enrichment of the original RDF document w.r.t. certain entailment regime. The user profile can also affect the interpretation of the RDF document as well as introduce implicit modifiers or filters. **User dynamical queries** will incorporate user profiles and make explicit such effects.

We have discussed queries that aim for different types of insights of the datasets. In query generation we would like to exploit such kinds of queries.

3.2 Query Parameters

As we mentioned at the beginning of this section, the control of query parameters is important for the comprehensibility of queries. Görlitz et. al recently presented the SPLODGE system [**?**] to generate benchmark queries based on systemic parameterisation of queries. Different from their intension,

we control query parameters to guarantee comprehensibility of queries. Following the introduction of SPARQL queries in Sec. **??** we consider the following parameters of queries in this deliverable:

1. **Query Type:** as discussed in the previous section, SELECT will be the focused query type from a query insightfulness point of view. Nevertheless, we are interested in other query types that users might use in reality;

2. **Modifier and Filter:** we are interested in the kind and frequency of different query modifiers and filters that uers might use to restrict or extend their results;

3. **Variable Number:** the number of variables in a query loosely determines how many matching is needed to obtain an answer as well as the comprehensibility of the query. Also, the number and combination of variables in a query triple (there are 8 possibilities in total) loosely determines how specific a query triple is;

4. **Triple Number:** the number of triples in a BGP loosely determines how many joining is needed to obtain all answers. More triples obvious make the query more lengthy and likely more difficult to read;

The above parameters will be considered when we investigate use cases.

4 Generated Queries for the Case Studies

In this section, we apply the framework described in Sec. **??** on datasets provided by case studies. The purpose of this study is to understand the factors considered by human users and experts when they generate queries for given datasets, so that we can apply similar principles in automatic query generation. Hence we will not look into the entire datasets used in our case studies, which is too big for manual investigation and should be analysed automatically. Instead, we look into realistically small datasets case studies.

We consider the public semantic data exploitation case study specified in Deliverable 1.1 [**?**] and healthcare case study specified in Deliverable 2.1 [**?**]. For each case study, we will present the following contents:

1. **Datasets Description:** a brief description of each dataset, its contents and basic statistics. We use $\mathcal{CN}$ to denote classes, i.e. resources used as objects of `rdf:type` predicate. We use $\mathcal{IN}$ to denote indivdiuals, i.e. resources used as subjects of `rdf:type` predicate. We use $\mathcal{RN}$ to denote object properties, i.e. predicates used between two individuals. We use $\mathcal{DN}$ to denote datatype properties, i.e. predicates used between an individual and a literal. We use $\mathcal{CA}$ to denote class assertion triples, i.e. triples with `rdf:type` as predicate. We use $\mathcal{RA}$ to denote object property assertion triples, i.e. triples with an object property as predicate. We use $\mathcal{DA}$ to denote datatype property assertion triples, i.e. triples with a datatype property as predicate. We use $\mathcal{AA}$ to denote annotation triples.

2. **Framework Application:** applying the framework presented in Sec. **??** on the datasets. Some analysis results are obtained with the help of domain experts;

3. **Query Generation:** based on the application of framework, we present several manually generated queries for each case study. For the sake of conciseness, we use `base` to represent all the namespaces (except the rdf and rdfs namespaces) and omit the prefixes. Note that they are not

meant to be all the queries that are generatable based on the framework, but rather to show how queries can be generated and what they will look like. A thorough generation will require an automatic generation mechanism;

4.1 Semantic Data Exploitation Case Study

4.1.1 Datasets Description

In this deliverable, we look into two datasets used in the semantic data exploitation case study. One is the Sports dataset, the other is the History dataset from DBPedia.

The Sports dataset is about sports events, players, organisations, especially in football and car racing. It directly or indirectly imports terminologies from 9 other datasets. A brief statistics about its importing closure is illustrated in Table **??**.

Table 1: Semantic Data Exploitation Case Study Statistics

Dataset	$\|\mathcal{CN}\|$	$\|\mathcal{RN}\|$	$\|\mathcal{DN}\|$	$\|\mathcal{IN}\|$	$\|\mathcal{CA}\|$	$\|\mathcal{RA}\|$	$\|\mathcal{DA}\|$	$\|\mathcal{AA}\|$
Sports dataset	76	96	63	4863	4816	3696	9775	6499
History dataset	4	0	0	10909	10933	0	0	30832

Several most populated classes of the Sport dataset include:

1. *FootballMatchEvent:* the instances of this class are, as the class name suggests, football match events. Each event usually associated to severl participants, and has a football match event type.
2. *FormulaOneEvent:* similar as above, the instances of this classes are F1 events. Each event has one or several participants.
3. *MediaFragment:* the instances of this class are media fragments of videos. Each fragment has datatype properties such as start time, finish time and description. Notedly, each media fragment is associated to some event and some video through annotation properties.
4. *SoccerPlayer:* the instances of this class are football players. Each player is usually associated to his/her current club through an annotation property, and may has several labels for his/her names.

The most referred to individuals usually belong to the following several classes:

1. *FootballMatchEventType:* although there are only 26 individuals of this classes, its instances are referred to by many football match events.
2. *FormulaOneRacer:* this class has 27 instances, many of which are frequently referred to as participants in F1 events.
3. *Video:* in the current dataset this class has only 12 instances but its instances are frequently referred to by media fragments.

The History dataset is about military conflicts. It does not import any other dataset. A brief statistics about it is illustrated in Table **??**. Different from the Sports dataset, large part of information in te History dataset is described with annotations. The four classes are:

1. *Combatant:* persons engaged in combats curing military conflicts. Instances of this classes are often referred to by military conflicts.

2. *MilitaryConflict:* instances of this class often refer to combatant, military persons as commanders, and populated places as places.

3. *MilitaryPerson:* instances of this class are often referred to as commanders by military conflicts;

4. *PopulatedPlace:* instances of this class are often referred to as places by military conflicts;

4.1.2 Framework Application

We apply the framework on the two datasets as follows:

Analysis	Sports dataset	History dataset
Nominative information	It is clear that *FootballMatchEvent* and *FormulaOneEvent* are two core topics of this dataset. Their participants are *SoccerPlayer* or *FormulaOneRacer*. They are recorded in the *MediaFragment*, which are part of *Video*. As we can see, *MediaFragment* associated to both events and videos, so they can also be regarded as topics of the dataset.	It is clear that *MilitaryConflict* is the core topic of this dataset. Their combatants are *Combatant*. Their commanders are *MilitaryPerson*. Their places are *PopulatedPlace*.
Topological information	There is a star-shaped structure around each event or each media fragment. There is also a chain-shaped structure that connects participants with events, media fragments and videos. There can be circle-shaped structures when multiple participants are involved in multiple same events. There is no obvious tree-shaped structures.	There is a star-shaped structure around each military conflict. There can be circle-sharped structures when multiple combatants/commanders/places associated to multiple same conflicts, and vice versa. There is no other obvious structures.
High correspondence	There is a high correspondence for soccer players of the same club to participant in same events. There is alos a high correspondence for media fragment of the same event to belong to the same video.	There is no obvious high correspondence query pairs.
Low correspondence	There is no obvious low correspondence query pairs.	
Exceptions	There can be exceptions of the high correspondence.	There is no obvious exception queries.
Temporal dynamics	This dataset does not have obvious temporal dynamics. Nevertheless, if more datasets of the similar kinds are collected, it is possible to discover temporal dyanmics of the current club of soccer players.	It is generally unlikely to have temporal dynamics in a history dataset, assuming the data collected are correct. Particularly, history dataset record historical facts, which should not subject to change.

continued ...

...continued

Analysis	Football dataset	History dataset
Spatial dynamics	The current dataset does not have obvious spatial dynamics. Nevertheless, if more datasets of the same kind are linked, they can potentially complement each to have more complete information about events and participants.	The current dataset does not have obvious spacial dynamics. Nevertheless, different history datasets may record the same military conflict from differetn perspective. When such datasets are linked, it is possible to discover dynamics such as casualties across different datasets.
Value dynamics	In this dataset, media fragments have time attributes. It is possible to have queries generated asking for longest/shortest media fragments. It is also possible to ask for players/races who have participated in most/least events.	It this dataset, it is possible to ask for combatants or military persons who have been involved in most/least military conflicts. It is also possible to ask for populated places in which most/least conflicts occured.
User dynamics	As mentioned above, different users may have different perspectives. It is interesting to present users with data that is most/least close to their perspectives, e.g. giving the *nearby* conflicts, or giving the players of the club in *my* city.	
Query type	According to our investigation with domain experts and users, SELECT query is mostly required.	
Modifier and filter	Depending on the type of query generated. For example, in exception query `FILTER NOT EXISTS` might be needed. In query for value dynamics modifiers such as `ORDER BY DESC` (ordering solutions in descending order) and `LIMIT` (restricting the number of solutions) will be used.	
Variable number	According to our investigation with domain experts and users, queries related to 2 to 3 entities are most required.	
Triple number	No specific requirement according to our domain experts or users.	

4.1.3 Query Generation

Based on the datasets and their analysis in previous subsections, we manually generate the following queries:

1. **Topic query:**
   - This query is about an football match event:

```
SELECT ?event ?type ?participant ?fragment
WHERE {
   ?event rdf:type base:FootballMatchEvent .
   ?event base:hasFootballMatchEventType ?type .
   ?event base:hasParticipant ?participant .
   ?fragment base:hasEvent ?event .}
```

   - This query is about a media fragment:

```
SELECT ?fragment ?event ?video ?start ?absoluteStart
       ?date ?description ?finish ?absoluteFinish
WHERE {
   ?fragment rdf:type base:MediaFragment .
   ?fragment base:hasEvent ?event .
   ?fragment base:isPartofVideo ?video .
   ?fragment base:startTimeCode ?start .
   ?fragment base:absoluteStartTimeCode ?absoluteStart .
   ?fragment base:startDate ?date .
   ?fragment base:description ?description .
   ?fragment base:finishtTimeCode ?finish .
   ?fragment base:absoluteFinishTimeCode ?absoluteFinish .}
```

- This query is about a military conflict:

```
SELECT ?conflict ?combatant ?label ?commander ?place
WHERE {
   ?conflict rdf:type base:MilitaryConfilict .
   ?conflict base:combatant ?combatant .
   ?conflict base:label ?label .
   ?conflict base:commander ?commander .
   ?conflict base:place ?place .}
```

2. **Structural query:** The topic queries above are all star-shaped queries. In addition, we have the following chain-shaped or circle-shaped query:

   - Chain-shaped:

```
SELECT ?event ?participant ?fragment ?video
WHERE {
   ?event base:hasParticipant ?participant .
   ?fragment base:hasEvent ?event .
   ?fragment base:isPartofVideo ?video .}
```

   - Circle-shaped:

```
SELECT ?event1 ?event2 ?participant1 ?participant2
WHERE {
   ?event1 base:hasParticipant ?participant1 .
   ?event1 base:hasParticipant ?participant2 .
   ?event2 base:hasParticipant ?participant1 .
   ?event2 base:hasParticipant ?participant2 .}
```

   - Circle-shaped:

```
SELECT ?conflict1 ?combatant1 ?conflict2 ?combatant2
WHERE {
   ?conflict1 base:combatant ?combatant1 .
   ?conflict1 base:combatant ?combatant2 .
   ?conflict2 base:combatant ?combatant1 .
   ?conflict2 base:combatant ?combatant2 .}
```

3. **Corresponding query:** The following two queries have high correspondences. Particularly, most of the solutions in the 1st query should be solutions of the 2nd query, indicating that two players of the same club should very likely be participants of the same event:

   - ```
     SELECT ?player1 ?player2
     WHERE {
        ?player1 base:currentClub ?club .
        ?player2 base:currentClub ?club .}
     ```
   - ```
     SELECT ?player1 ?player2
     WHERE {
        ?event base:hasParticipant ?player1 .
        ?event base:hasParticipant ?player2 .
        ?player1 base:currentClub ?club .
        ?player2 base:currentClub ?club .}
     ```

4. **Exeption query:** The following exception query is associated to the above corresponding query, i.e. players of the same club but never participate in the same event:

   ```
   SELECT ?player1 ?player2
   WHERE {
      ?player1 base:currentClub ?club .
      ?player2 base:currentClub ?club .
      FILTER NOT EXISTS {?event base:hasParticipant ?player2 .
         ?event base:hasParticipant ?player1 .} }
   ```

5. **Value dynamical query:**

   - This query asks for the top 20 media fragments with latest starting time:

     ```
     SELECT ?fragment ?absoluteStart
     WHERE {
        ?fragment base:absoluteStartTimeCode ?absoluteStart .}
        ORDER BY DESC ?absoluteStart .
        LIMIT 20 .
     ```

   - This query asks for the top 20 combatants who have fought in most military conflicts:

     ```
     SELECT ?combatant (COUNT(?conflict) AS ?count)
     WHERE {
        ?conflict base:combatant ?combatant .}
        ORDER BY DESC(?count) .
        LIMIT 20 .
     ```

4.2 Healthcare Case Study

4.2.1 Datasets Description

Now we look into a dataset used in the healthcare case study. This is a de-identified patient dataset. It contains medical records of patients. It does not import any other dataset. A brief statistics about

Table 3: Healthcare Case Study Dataset Statistics

Dataset	$\lvert\mathcal{CN}\rvert$	$\lvert\mathcal{RN}\rvert$	$\lvert\mathcal{DN}\rvert$	$\lvert\mathcal{IN}\rvert$	$\lvert\mathcal{CA}\rvert$	$\lvert\mathcal{RA}\rvert$	$\lvert\mathcal{DA}\rvert$	$\lvert\mathcal{AA}\rvert$
Patient dataset	6	0	0	15283	15283	0	0	173759

it is illustrated in Table **??**. Similar as the History dataset, large part of its information is encoded in annotation properties.

The six classes of this dataset are as follows. It's worth mentioning that although an observation or an procedure occurs in an encounter, which all have documents, their documents can be different:

1. *Document:* instances of this classes are documents of encounters, documents, procedures. They are referred to by instances of those classes;
2. *Encounter:* instances of this classes are events in which observations and procedures occur;
3. *Patient:* instances of this classes are patients. Their EMPI IDs are referred to by instances of observations and procedures;
4. *Observation:* as name suggests, instances of this class are observation on patients;
5. *Procedure:* similar as above, instances of this class are procedures on patients;
6. *SubstanceaAdministrator:* this class is associated to some encounter, patient and document.

4.2.2 Framework Application

We apply the framework on the two datasets as follows:

Analysis	Patient dataset
Nominative information	In this healthcare domain dataset, there can be different perspectives regarding the central topic of the dataset, which can be *Document*, *Encounter*, *Patient*, or all of them. In fact, *Observation*, *Procedure* and *SubstanceAdministrator* are all associated to *Document*, *Patient* and *Encounter*, which is also associated to *Document*.
Topological information	There is an obvious star-shaped structure around each *Observation*, *Procedure* and *SubstanceAdministrator* as they are related to *Document*, *Patient* and *Encounter*. There is also a chain-shaped structure that connects *Patient* with *Observation*, *Procedure* or *SubstanceAdministrator*, then *Encounter* and *Document*. There can be circle-shaped structures revolving *Encounter*. Particularly, *Observation*, *Procedure* and *SubstanceAdminstrator* and their *Encounter* can have different *Documment*, but all these documents refer to the same *Encounter*. There is an obvious tree-shaped structure revolving *Patient*. Particularly, each patient can be involved in multiple encounters, which can involve multiple observations, procedures and substance administrators, which all have their documents.

continued ...

...continued

Analysis	Patient dataset
High correspondence	There is a high correspondence for documents of *Observation*, *Procedure* and *SubstanceAdministrator* that occur in the same *Encounter* to refer to the same *Encounter*. Another interesting correspondence, according to our domain experts, is that same *Observation* are frequently occur on related *Patient*. For example, if a patient has a family history of diabetes, then diabetes will be more likely to occur on this patient. Nevertheless, our test dataset alone does not include family information of patients.
Low correspondence	There is no obvious low correspondence query pairs.
Exceptions	There can hardly be any exception of the high correspondence on *Encounter*. Any retrieved exception should be considered as an indicator of errorneous data. Exception on the diabetes correspondence may also indicate lack of examination.
Temporal dynamics	There is a temporal dynamics on the values of observations. For example, for an observation with a numerical value, its average, maximal, minimal value in the last 3 months are temporally dynamic.
Spatial dynamics	The current dataset does not have obvious spatial dynamics. Nevertheless, if more datasets of the same kind are linked, data from different medical facilities, such as blood pressure values from medical home and those from hospitals, can be different from each other.
Value dynamics	For *Observation* with numeric values, it is possible to ask those with highest/lowest values. It is also possible to ask for *Document*, *Encounter*, *Observation*, etc., with earliest/latest effective time.
User dynamics	According to our domain experts, queries submited by physicians and patients may have different results, for example, to query the latest HbA1c lab test value, physicians may only regard those values within three months as valid, while those values beyond threee months as invalid, due to the effective time of HbA1c is three-month in a healthcare perspective.
Query type	According to our investigation with domain experts, if the query is to evaluate some clinical expression, then `ASK` query should be used. If the query is to retrieve some patients, then `SELECT` query should be used.
Modifier and filter	In addition to the modifiers and filters used in the previous case study, `UNION` is also quite often used to union results satisfy different constraints.
Variable number	According to our investigation with domain experts and users, 1-5 variables, such as WHO has WHICH observation and used WHICH drug, are most required.
Triple number	According to our investigation with domain experts and users, 1-5 triples are most required.

4.2.3 Query Generation

Based on the datasets and their analysis in previous subsections, we manually generate the following queries:

1. **Topic query:**

   - This query is about a patient:

```
SELECT ?patient ?observation ?document ?birthDay ?gender
WHERE {
   ?document base:OPTIM_DOCUMENT_EMPI_ID ?id .
   ?patient base:OPTIM_PATIENT_EMPI_ID ?id .
   ?patient base:OPTIM_PATIENT_BIRTH_DATE ?birthDay .
   ?patient base:OPTIM_PATIENT_GENDER ?gender .
   ?observation base:OPTIM_OBSERVATION_EMPI_ID ?id .}
```

   - This query is about an observation:

```
SELECT ?patient ?observation ?document ?encounter ?value
WHERE {
   ?patient base:OPTIM_PATIENT_EMPI_ID ?id .
   ?observation base:OPTIM_OBSERVATION_EMPI_ID ?id .
   ?observation base:OPTIM_OBSERVATION_DOCUMENT_ID ?document .
   ?observation base:OPTIM_OBSERVATION_ENCOUNTER_ID ?encounter .
   ?observation base:OPTIM_OBSERVATION_VALUE_NUMERIC ?value .}
```

2. **Structural query:** In addition to the star-shaped query above, examples of other structural queries are:

   - Chain-shaped:

```
SELECT ?patient ?observation ?encounter ?document
WHERE {
   ?patient base:OPTIM_PATIENT_EMPI_ID ?id .
   ?observation base:OPTIM_OBSERVATION_EMPI_ID ?id .
   ?observation base:OPTIM_OBSERVATION_ENCOUNTER_ID ?encounter .
   ?encounter base:OPTIM_ENCOUNTER__DOCUMENT_ID ?document .}
```

   - Circle-shaped:

```
SELECT ?observation ?encounter ?document
WHERE {
   ?observation base:OPTIM_OBSERVATION_DOCUMENT_ID ?document .
   ?observation base:OPTIM_OBSERVATION_ENCOUNTER_ID ?encounter .
   ?document base:OPTIM_DOCUMENT_ENCOUNTER_ID ?encounter .}
```

   - Tree-shaped:

```
SELECT ?patient ?encounter ?observation ?procedure
WHERE {
   ?patient base:OPTIM_PATIENT_EMPI_ID ?id .
   ?observation base:OPTIM_OBSERVATION_EMPI_ID ?id .
   ?observation base:OPTIM_OBSERVATION_ENCOUNTER_ID ?encounter .
   ?encounter base:OPTIM_ENCOUNTER__DOCUMENT_ID ?document .}
   OPTIONAL { ?procedure base:OPTIM_PROCEDURE_EMPI_ID ?id .
      ?procedure base:OPTIM_PROCEDURE_ENCOUNTER_ID ?encounter .}
```

3. **Corresponding query:** The following two queries have high correspondences. Particularly, most of the solutions in the 1st query should be solutions of the 2nd query, indicating that the encounter of a document should be the same as the encounter of the observation that the document is about:

   - ```
     SELECT ?encounter ?document
     WHERE {
        ?document base:OPTIM_DOCUMENT_ENCOUNTER_ID ?encounter .}
     ```
   - ```
     SELECT ?encounter ?document
     WHERE {
        ?observation base:OPTIM_OBSERVATION_DOCUMENT_ID ?document .
        ?observation base:OPTIM_OBSERVATION_ENCOUNTER_ID ?encounter .}
     ```

4. **Exception query:** The following exception query is associated to the above corresponding query, i.e. an observation and its document having different encounters:

   ```
   SELECT ?encounter ?document
   WHERE {
      ?observation base:OPTIM_OBSERVATION_DOCUMENT_ID ?document .
      ?observation base:OPTIM_OBSERVATION_ENCOUNTER_ID ?encounter .
      FILTER NOT EXISTS
      {?document base:OPTIM_DOCUMENT_ENCOUNTER_ID ?encounter} .}
   ```

5. **Value dynamical query:**

   - This query asks for the top 20 observations with highest numeric values:

     ```
     SELECT ?observation ?value
     WHERE {
        ?observation base:OPTIM_OBSERVATION_VALUE_NUMERIC ?value .}
        ORDER BY DESC ?value .
        LIMIT 20 .
     ```

   - This query asks for the maximal observation value of each patient that is observed after Jan 1st, 2010:

     ```
     SELECT ?patient (MAX(?value) AS ?max)
     WHERE {
        ?patient base:OPTIME_PATIENT_EMPI_ID ?id .
        ?observation base:OPTIM_OBSERVATION_EMPI_ID ?id .
        ?observation base:OPTIM_OBSERVATION_VALUE_NUMERIC ?value .
        ?observation base:OTIME_OBSERVATION_EFFECTIVETIME_HIGH ?time .
        FILTER (?time > "2010-01-01T00:00:00Z"^^xsd:dateTime ) .}
     ```

5 Conclusion

This deliverable investigates the problem of query generation in the context of the case studies in K-Drive project. To address this problem, this deliverable focus on the construction of a query generation framework and its manual application on several datasets collected from case studies.

Our framework is based on the idea that generated queries should be insightful and comprehensible to dataset users. To exploit the insights, we look into the nominative and topological information in datasets, the relation between different subsets of the datasets, and the dynamics of the datasets and queries. To improve comprehensibility, we also control the parameters of the queries generated.

As a result of this study, we present analysis of case study datasets with help from domain experts in usecase partners. We also present a set of generated queries for the case studies. According to our investigation, datasets usually have obvious central topics, whose properties an attributes normally constitute star-shaped queries. Other topological structures are also found in datasets. Groups of sub-graphs having high correspondence are more frequently found in datasets than those with low correspondence. Temporal and spatial dynamics are hard to observe in a single dataset but are quite frequent in application scenario where data are collected continuously and distributively. Value dynamics happens often when datasets contain datatype properties and literals. User dynamics happens when datasets are expected to be accessed with users having different perspectives. In most applications, `SELECT` queries are used by users. Modifiers and filters such as `FILTER NOT EXISTS`, `ORDER BY DESC`, `LIMIT` are frequently used. Different applications have different levels of control on number of variables and triples in queries.

Based on these analysis, different kind of insightful queries are manually generated.

In the next step of the collaboration in this work package, we will develop automatic generation technologies based on our framework and analysis so that to improve the efficiency and coverage of generated queries.

Acknowledgement

This research has been funded by the European Commission within the 7th Framework Programme/Maria Curie Industry-Academia Partnerships and Pathways schema/PEOPLE Work Programme 2011 project K-Drive number 286348 (cf. `http://www.kdrive-project.eu`).