Fábio Gomes dos Santos

PROLOG VERSUS XQUERY PROCESSORS: A PERFORMANCE EVALUATION OF XML QUERIES PROCESSING METHODS

Dissertation presented to the Computing Graduate program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master of Science. Research area: Software Engineering.

Advisor: Prof. D.Sc. Vanessa Braganholo Murta

Niterói

2015

FÁBIO GOMES DOS SANTOS

PROLOG VERSUS XQUERY PROCESSORS: A PERFORMANCE EVALUATION OF XML QUERIES PROCESSING METHODS

Dissertation presented to the Computing Graduate program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master of Science. Research area: Database and Structured Documents.

Approved on July 2015.

APPROVED BY

Prof. D.Sc. Vanessa Braganholo Murta – Advisor

IC-UFF

Prof. D.Sc. Carina Friedrich Dorneles

INE-UFSC

Prof. D.Sc. Daniel Cardoso Moraes de Oliveira

IC-UFF

Prof. D.Sc. Aline Marins Paes Carvalho

IC-UFF

Niterói

2015

ACKNOWLEDGMENTS

I would like to thank my grandparents, parents, brother, sister and fiancée for supporting my decisions. This work would not be possible without their support.

I would also like to express the deepest appreciation to my professors Vanessa and Aline, who have guided my studies and continually gave me inspiration.

In addition, I thank my graduate and postgraduate computer science classmates for the many times that we got together to study. Certainly, these situations helped me to reach this point.

RESUMO

XML tem sido amplamente utilizado para representar os dados na web. Com tanta informação armazenada em documentos XML, a necessidade de extrair informações destes documentos aumenta. Existem muitas abordagens para consultar esses documentos. No entanto, elas estão preocupadas apenas em extrair informações explícitas contidas neles. Além disso, as abordagens atuais nem sempre têm um desempenho satisfatório quando aplicadas a grandes volumes de dados. Neste trabalho, propomos avaliar as abordagens alternativas que propõem extrair informações de documentos XML usando Prolog. As avaliações realizadas usando essas abordagens surpreendem por mostrar que elas, muitas vezes, fornecem melhores tempos de resposta para o processamento de consultas quando comparados com as abordagens nativas XML, mesmo quando as consultas não envolvem o uso de inferência. No entanto, em que situações isso ocorre? O objetivo desse trabalho é tentar responder a essa pergunta. Para isso, avaliamos o desempenho de um conjunto extenso de consultas e as executamos com processadores XQuery e Prolog, a fim de medir o desempenho. Ao final, propomos um conjunto de heurísticas que podem ser utilizadas para avaliar em que situações é vantajoso utilizar Prolog para executar consultas XQuery.

Palavras-chave: XML, inferência, Prolog, benchmark, extração de informação.

ABSTRACT

XML has been widely used to represent data on the web. With so much information stored in XML documents, the necessity to extract information from these documents arises. There are many approaches to query these documents. However, they are concerned only in extracting explicit information contained therein. Furthermore, the current approaches do not always have satisfactory performance when applied to large volumes of data. In this work we evaluate alternative approaches that extract information from XML documents using Prolog. The evaluation made using these approaches surprisingly shows that, many times, they provide better processing time results when compared to other native XML querying approaches. However, in what kind of situation this happens? This work aims at answering this question. To achieve this goal, we evaluate the performance of an extensive set of queries and execute it using XQuery and Prolog engines in order to evaluate the performance. Finally, we propose a set of heuristics that can be used to evaluate situations where it is advantageous to use Prolog to execute XQuery queries.

Keywords: XML, inference, Prolog, benchmark, extraction of information.

LIST OF FIGURES

Figure 1: Evaluation workflow 13
Figure 2: Overview of Lima (2012) 16
Figure 3: Example of XML document 17
Figure 4: Example of Prolog translation 18
Figure 5: XML example (ALMENDROS-JIMÉNEZ; BECERRA-TERÓN; ENCISO-BANOS, 2008) 20
Figure 6: Prolog translation by Almendros *et al*. (2008) 21
Figure 7: Prolog translation by Lima (2012) 21
Figure 8: Comparing XQuery with a Prolog query 24
Figure 9: Shell Script to call Sedna and get time results 24
Figure 10: Shell script to compute Standard deviation and Mean 25
Figure 11: XBench TC/MD diagram 26
Figure 12: Example of XQuery from XBench 27
Figure 13: Translated Prolog query 27
Figure 14: XQuery example using *where* 27
Figure 15: Translated Prolog query from XQuery in Figure 14 28
Figure 16: XQuery example using *order by* 28
Figure 17: Translated Prolog query from XQuery in Figure 16 29
Figure 18: Execution flow 31
Figure 19: Comparing SWI versus YAP execution times – DC/SD 35
Figure 20: Comparing SWI versus YAP execution times – TC/SD 36
Figure 21: Comparing SWI versus YAP execution times – TC/MD 36
Figure 22: Comparing Sedna versus eXist execution times – DC/SD 37
Figure 23: Comparing Sedna versus eXist execution times – TC/SD 38
Figure 24: Comparing Sedna versus eXist execution times – TC/MD 38
Figure 25: Comparing Sedna versus SWI execution times – DC/SD 39
Figure 26: Comparing Sedna versus SWI execution times – TC/SD 40
Figure 27: Comparing SWI versus eXist execution times – TC/MD 41
Figure 28: Comparing SWI, YAP, Sedna and eXist execution times – DC/SD 43
Figure 29: Comparing SWI, YAP, Sedna and eXist execution times – TC/SD 43
Figure 30: Comparing SWI, YAP and Sedna execution times – TC/MD 44
Figure 31: Element definition that causes a sorting triggered by schema 46

Figure 32: Desired workflow 53

LIST OF TABLES

Table 1: Query Mechanisms Execution time for DC/SD (in seconds) 32
Table 2: Query Mechanisms Execution time for TC/SD (in seconds) 33
Table 3: Query Mechanisms Execution time for TC/MD (in seconds) 34
Table 4: Statistical significance for Sedna versus SWI execution times – DC/SD 40
Table 5: Statistical significance for Sedna versus SWI execution times – TC/SD 41
Table 6: Statistical significance for Sedna versus SWI execution times – TC/MD 42
Table 7: Technical features for data centric single document 46
Table 8: Technical features for text centric single document 47
Table 9: Technical features for text centric multiple document 48
Table 10: Heuristic map 49

LIST OF ACRONYMS AND ABBREVIATIONS

DC – Data Centric

MD – Multiple Document

SD – Single Document

TC – Text Centric

XBench – A XML benchmark

XML – Extensible Markup Language

XPath – XML Path language

TABLE OF CONTENTS

Chapter 1 – Introduction 12

Chapter 2 – Using Prolog to Represent and Query XML Documents 15

2.1 XMLInference 15

2.2 Almendros-Jiménez *et al.* 20

2.3 Running XML Queries in Prolog 22

Chapter 3 – Experimental Setup 23

3.1 Overview 23

3.2 Execution Scripts 23

3.3 Translating XML Queries TO Prolog Queries 26

3.4 Query Mechanisms 29

3.5 Setting Environment 30

Chapter 4 – An Heuristic for XML Query Processing 32

4.1 Overview 32

4.2 Results 32

4.2.1 Prolog engines 35

4.2.2 Native XML tools 37

4.2.3 XML Native Tools versus SWI 39

4.2.4 Comparing All Approaches 43

4.3 Discussion 44

Chapter 5 – Conclusion 51

5.1 Final Remarks 51

5.2 Future Work 52

References 55

Appendix I – XML Schemas of Datasets 57

1.1 TC/SD SCHEMA 57

1.2 TC/MD SCHEMA 60

1.3 DC/SD SCHEMA 64

Appendix II – Benchmarck Queries 71

1.1 Single Document Data Centric XQuery Queries 71

1.2 Single Document Text Centric XQuery Queries 93

1.3 Multiple Document Text Centric XQuery Queries 118

CHAPTER 1 – INTRODUCTION

XML has quickly become a universal standard for information exchange over the Web. The rise of large data collections in this format puts in evidence the need of improving query methods, so that large portions of data can be efficiently managed (KOTSAKIS, 2002).

Most of the XML query languages currently available, such as XPath (CLARK; DEROSE, 1999) and XQuery (BOAG *et al.*, 2010), gather the query answers by navigating from the root of the document to an internal element of interest, applying filters to obtain the desired result. The query is usually processed over the explicit data, ignoring any implicit information that may be obtained from it. This implicit information, however, is crucial in many applications and because of this, alternative approaches using Prolog were proposed (ALMENDROS-JIMÉNEZ; BECERRA-TERÓN; ENCISO-BANOS, 2008; LIMA, 2012) aiming to provide ways to get more information from XML datasets. Consider, for instance, provenance data collected from workflow executions (FREIRE *et al.*, 2008; MATTOSO *et al.*, 2010). In this scenario, there is a lot of implicit information that cannot be easily queried, such as "what were the activities that contributed to produce a given workflow output?".

Although these alternative approaches provides new ways to get information from XML datasets, their main focus is on how to infer data from XML documents using Prolog engines. However, some works (ALMENDROS-JIMÉNEZ; BECERRA-TERÓN; ENCISO-BANOS, 2008; LIMA, 2012) show that some queries were faster when executed in Prolog than in XQuery engines. However, there is no study on the impacts of the adoption of this approach to run a broader set of XQuery queries. What kind of queries benefit from their approach? Can we always use Prolog to execute XQuery in an efficient way? There are many questions involved in adopting an alternative approach to run XQuery. In this work, we investigate this matter. More specifically, we focus in queries execution times of both kinds of approaches (XQuery native engines versus Prolog query engines).

Our methodology consists in evaluating a set of XML query processing engines and comparing their results. The goal is to delineate a set of heuristics that can be used to decide if it is worth to use Prolog to process a given XML query.

To evaluate the different query processing engines in an organized and fair way, we created and followed the workflow shown in Figure 1. It uses XML documents and XQuery queries provided by the XBench benchmark (YAO; OZSU; KHANDELWAL, 2004). XBench provides different types of datasets. The dataset can be composed of a single document (SD) or multiple documents (MD) and can be data centric (DC) or text centric (TC). Over these

datasets, we execute the set of queries provided by XBench using the following query processing engines: Galax (SIMÉON *et al.*, 2000) (a command line XQuery processor), Sedna (FOMICHEV; GRINEV; KUZNETSOV, 2006) and eXist (MEIER, 2003) (two native XML DBMS), and two different Prolog engines, SWI (JAN WIELEMAKER *et al.*, 2012) and Yap (COSTA; ROCHA; DAMAS, 2012). However, in order to use the Prolog engines, we must first translate the XML document into Prolog facts using one of the existing approaches (ALMENDROS-JIMÉNEZ; BECERRA-TERÓN; ENCISO-BANOS, 2008; LIMA, 2012).

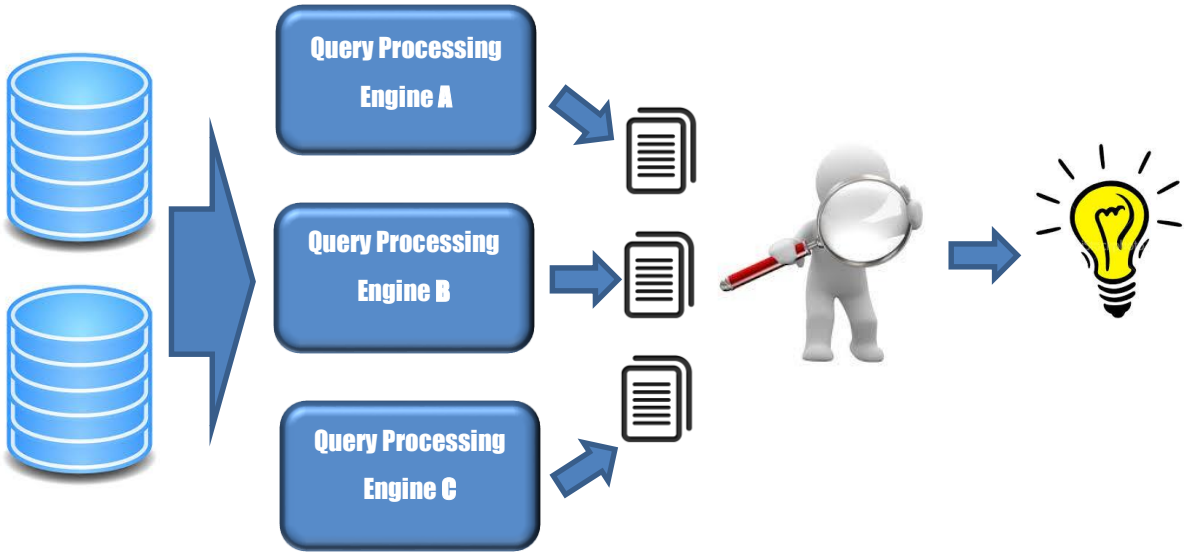


Figure 1: Evaluation workflow

The main challenge in this evaluation is on how to translate the XQuery queries into equivalent Prolog queries. In fact, we did not find any work that provides a translation algorithm. However, before investing time in a translation algorithm, which would be very complex due to the diversity of XQuery constructions, we must analyze the query processing performance to decide if the effort would pay off. This is the main objective of this dissertation. Thus, in this work, XQuery queries are translated to Prolog in an ad-hoc manner. To proceed with our evaluation, we gather the query processing time for each query. Finally, we evaluate the results and propose a set of heuristics that can be used to choose the better approach to process a given type of query.

Our evaluation is guided by the following research questions (RQs) that are answered throughout this work:

RQ1: Can we represent the XQuery queries proposed by XBench as standard Prolog queries and still get the same results?

RQ2: Do the translated queries in RQ1 present a better or competitive execution time when compared to native XML query processing engines?

RQ3: When is it a good choice to use Prolog-based approaches to process XQuery queries?

The remaining of this work is organized in five chapters. Chapter 2 presents approaches to convert XML into Prolog facts. Chapter 3 explains how our evaluation approach was set up based on the workflow shown in Figure 1 and explains how the results were gathered. In Chapter 4, we describe the query processing engines that were used in this task and show the results gathered by each query processing engine. We also present our heuristics in Chapter 4. Finally, Chapter 5 concludes this work.

CHAPTER 2 – USING PROLOG TO REPRESENT AND QUERY XML DOCUMENTS

There is several work in the literature that present benchmarks for comparing native XML approaches against alternative approaches (BÖHME; RAHM, 2001; RUNAPONGSA *et al.*, 2006; SCHMIDT *et al.*, 2001, 2002; YAO; OZSU; KHANDELWAL, 2004). Additionally, there are some approaches that propose mechanisms to convert XML to Prolog facts and rules aiming to query the content using Prolog engines (ALMENDROS-JIMÉNEZ; BECERRA-TERÓN; ENCISO-BANOS, 2007, 2008; LIMA, 2012; LIMA *et al.*, 2012a, b; SANTOS; PINHEIRO; BRAGANHOLO, 2012). However, to the best of our knowledge, our work is the first to propose an heuristic to evaluate when it is a good option use Prolog query mechanism to process XQuery, and when it is not.

In this chapter we discuss some works that present alternative approaches to query XML datasets using Prolog. Section 2.1 shows XMLInference (LIMA *et al.*, 2012b), the approach we use to translate XML to Prolog facts in our experiments. Section 2.2 presents an alternative approach to convert XML into Prolog facts. Finally, Section 2.3 discusses work that translates XML queries into Prolog.

2.1 XMLINFERENCE

The approach proposed by Lima *et al*. (2012) processes data contained in XML databases with the goal of inferring explicit and implicit information from it. It enables more complex answers to be obtained from either explicit information or information deducted through inference rules (implicit information). To make this possible, their work builds a knowledge base with facts (representing the XML data), rules (derived from the document's schema), and manual rules (provided by an expert user). This extends the query possibilities, providing the means for inference of new information.

Their approach is named XMLInference and is divided in three stages: rules configuration, generation of prolog facts, and query. Figure 2 illustrates each stage and its phases. To configure rules, in the first stage, the user uploads the XML Schemas used to validate the XML documents that will be queried. The schemas are analyzed by a schema translator component that generates Prolog rules from the processed data. These rules are called "automatic rules". This component is also responsible for generating predicate definitions, which are Prolog predicate signatures. Using these definitions, a specialist user can manually generate new rules. In the end of the first stage, the knowledge base has Prolog

rules that were automatically derived by the XMLInference approach, and rules that were added by the specialist user. This knowledge base can be reused whenever needed. It is important to note that the automatic rules are not essential, and that an XML document without a schema can also be queried based solely on the structure of the facts.

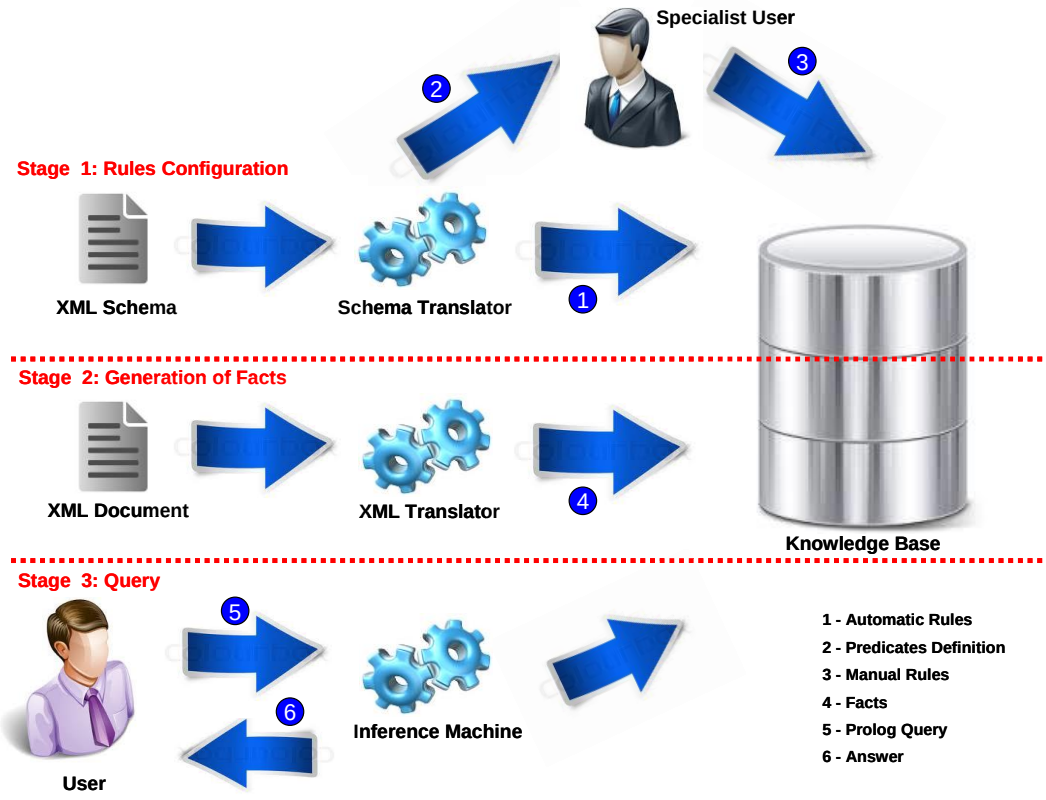


Figure 2: Overview of Lima (2012)

The second stage uses the XML translator component to derive Prolog facts from the XML documents. Finally, in the third stage, the user can execute queries that are processed by the Prolog inference machine. The results are presented to the user that may submit additional queries, since the environment is all set up.

Next, we discuss and show how XML Inference translates XML documents into Prolog facts (stage 2 of Figure 2). Since our *ad-hoc* query translation do not use the rules generated by XMLInference, they are not shown in this chapter.

The translation process generates several facts for a single XML document. It transforms XML elements into predicates, and its content into constants. Two identifiers (Prolog constants) are added to the facts. The first identifier is called *parent ID*, and is used to relate distinct predicates, so that facts that represent parent/child elements in the XML document can be connected. The second identifier is called *element ID*, and it used as a *parent*

ID by its children. Because of this, in the original approach (LIMA *et al.*, 2012b), only internal elements would have an *element ID* (since leaf elements do not have children). However, this id can also be used to keep the order of elements of the same parent when reconstructing the XML document.

```
01. <article>
02.    <publishInfo>
03.      <libId>000001</libId>
04.    </publishInfo>
05.    <institution>
06.        <university>Universidade Federal Fluminense</university>
07.      </institution>
08.    <title>
09.      <name>Towards ...</name>
10.      <subtitle>Prolog x XQuery</subtitle>
11.    </title>
12.    <prolog>
13.       <authors>
14.          <author>Richard Williams</author>
15.          <coauthor>Howard Mike</coauthor>
16.       </authors>
17.    </prolog>
18.    <body>
19.       <abstract><p>In this work ...</p></abstract>
20.       <section heading="Introduction"></section>
21.    </body>
22.    <epilog>
23.       <acknowledgments><p>I woud like ...</p></acknowledgments>
24.       some other text goes here ...
25.    </epilog>
26.    <notes owner="Rebeca">It must correct ...</notes>
27.    <size>
28.      <numberOfPages>125</numberOfPages>
29.      <bytes>16232</bytes>
30.    </size>
31.    <references>
32.      <author>Fabio</author>
33.      <paperTitle>Evaluating ...</paperTitle>
34.    </references>
35.    .
36.    .
37.    .
38. </article>
```

Figure 3: Example of XML document

When processing queries, order is an important matter. In fact, in our initial experiments, we noticed that some query output results didn't match that of other (native XML) approaches. Comparing these output results, we could see that the problem resided in the order of the leaf elements in the result. As an example, suppose an XML document with root *A* and children leaf elements *B* and *C*. When we query for *A*, elements *B* and *C* should appear inside *A*, in this order (*B* followed by *C*). In our query results, however, sometimes *C*

would appear before *B* inside *A*. Because of this, we had to add the *element ID* (which we sometimes call *order ID*) in leaf nodes as well. In this work, we use this modified version of Lima's approach instead of the original one – the second identifier is used to sort the child leaf elements so that their order is preserved in the query results.

The XML to Prolog translation process is guided by five possible translation types, as follows. An example of the application of these translations is presented in Figure 3 and Figure 4. This example will be used throughout this section.

```
01. article(1).
02. publishinfo(1, 2).
03. libid(2, 3, '000001').
04. institution(1, 5).
05. university(5, 6, 'Universidade Federal Fluminense').
06. title(1, 8).
07. name(8, 9, 'Towards ...').
08. subtitle(8, 11, 'Prolog x XQuery').
09. prolog(1, 13).
10. authors(13, 14).
11. author(14, 15, 'Richard Williams').
12. coauthor(14, 17, 'Howard Mike').
13. body(1, 19).
14. abstract(19, 20).
15. p(20, 21, 'In this work ...').
16. section(19, 23).
17. heading(23, 24, 'Introduction').
18. epilog(1, 26).
19. acknowledgments(26, 27).
20. p(27, 28, 'I woud like ...').
21. xml/mixedElement(26, 30, 'some other text goes here ...').
22. notes(1, 31, 'It must correct ...').
23. owner(31, 32, 'Rebeca').
24. size(1, 35).
25. numberofpages(35, 36, '125').
26. bytes(35, 38, '16232').
27. references(1, 40).
28. author(40, 41, 'Fabio').
29. papertitle(40, 43, 'Evaluating ...').
```

Figure 4: Example of Prolog translation

Translation of the document root. The root element of an XML document should be treated differently from the remaining ones because it has a characteristic that the others do not. It is the only element that has no parent. In our example of XML file (Figure 3), we can see on line 1 an example of an XML root element. Except for some very particular cases, the root element is a complex element. As shown in line 1 of Figure 4, the translation of a complex root element will generate a Prolog fact with the name of that element and an ID argument

generated by the application. The role of this ID is to connect the child elements with the root element.

Translation of complex elements. A complex element can be defined as an XML element that contains other elements and/or attributes as children. We can see an example in line 2 of Figure 3. To translate this element, XMLInference generates a fact with the name of respective complex element and two arguments. The first argument is the parent's ID and second one is the element's ID, as shown in the example of line 2 in Figure 4. Both IDs are used with the same objective. XMLInference generates them to relate the children elements with their parent. The children elements are translated accordingly to their types (simple, complex, etc.).

Translation of leaf elements with no attributes. A leaf element with no attributes has its textual content as its single child in the XML tree (see Figure 3, line 3). Such elements are translated as Prolog facts named after the element’s name. In this case, the fact has the parent ID, the element ID (which can be used for sorting), and the element content. (Figure 4, line 3).

Translation of leaf elements with attributes. Attributes are considered children of the element that contains them (Figure 3, line 26). XMLInference generates a fact that represents the element (Figure 4, line 22), plus a fact for each of its attributes (Figure 4, line 23). The former has the element name and three arguments: the parent’s ID, the element’s ID and its textual content. The remaining facts have the attribute name and three arguments: the parent’s ID (that represents the element that contains the attribute), the attribute ID (for sorting purposes) and the attribute value. Note that, although attributes are not ordered in the XML model, we decided to include an attribute ID for attributes so that we can reconstruct the original XML document exactly as it was.

Translation of mixed elements. XMLInference translates XML mixed elements in a way very similar to the way complex elements are translated. An important difference is the generation of facts named "xml/mixedElement" to represent the element's text children. This is needed because they have no associated name. The association of the name "xml/mixedElement" does not compromise the approach once the choice took into account the specification of the XML language. This specification does not allow the existence of XML element names with the character "/". This way, we can ensure that the translation

process of XML elements will not generate a prolog fact called "xml/mixedElement". Mixed elements also need and element ID for sorting purposes. This way, we generate “xml/mixedElement” facts with three arguments: the parent’s ID (the complex element), the element ID (for sorting purposes) and its textual content (Figure 4, line 21). The remaining children are translated accordingly to their types.

We chose XMLInference (LIMA *et al.*, 2012b) approach as our tool to convert XML documents into Prolog facts. In XMLInference, the authors have presented an evaluation where native XML approaches are compared against Prolog query mechanisms. Their experiments were executed with few test cases and their work do not propose a heuristic like our work does. However, because of the good results achieved by the XMLInference approach (LIMA, 2012), in this work we deepen the analysis to find when it is a good option to use Prolog query mechanisms to process XML queries, or when using native XML approaches is a better option.

2.2 ALMENDROS-JIMÉNEZ *ET AL.*

Almendros-Jiménez *et al*. (2008) represent XML documents by means of a logic program. This work is similar to Lima’s work (2012). Although there are a few differences on the Prolog facts translation, both approaches convert schema into rules and elements into facts, and elements are linked by IDs in the prolog facts.

```
<books>
  <book year="2003">
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
    <title>Data on the Web</title>
    <review>A <em>fine</em> book.</review>
  </book>
  <book year="2002">
    <author>Buneman</author>
    <title>XML in Scotland</title>
    <review><em>The <em>best</em> ever!</em></review>
  </book>
</books>
```

Figure 5: XML example (ALMENDROS-JIMÉNEZ; BECERRA-TERÓN; ENCISO-BANOS, 2008)

We can see the differences between these approaches in Figure 6 and Figure 7. Figure 5 shows an example of a simple XML document. Figure 6 and Figure 7 show the Prolog

translation result for Lima's work (2012a, b) and Almedros' work (2008), respectively. Note that Almendros uses lists, while Lima uses simple tokens. These lists are responsible for linking the elements and they are named **nodenumber**.

```
year('2003', [1, 1], 3).
author('Abiteboul', [1, 1, 1], 3).
author('Buneman', [2, 1, 1], 3).
author('Suciu', [3, 1, 1], 3).
title('Data on the Web', [4, 1, 1], 3).
unlabeled('A', [1, 5, 1, 1], 4).
em('fine', [2, 5, 1, 1], 4).
unlabeled('book.', [3, 5, 1, 1], 4).
year('2002', [2, 1], 3).
author('Buneman', [1, 2, 1], 3).
title('XML in Scotland', [2, 2, 1], 3).
unlabeled('The', [1, 1, 3, 2, 1], 6).
em('best', [2, 1, 3, 2, 1], 6).
unlabeled('ever!', [3, 1, 3, 2, 1], 6).
```

Figure 6: Prolog translation by Almendros *et al*. (2008)

```
books(1).
book(1, 2).
year(2, 3, '2003').
author(2, 5, 'Abiteboul').
author(2, 7, 'Buneman').
author(2, 9, 'Suciu').
title(2, 11, 'Data on the Web').
review(2, 13).
xml/mixedElement(13, 14, 'A ').
em(13, 15, 'fine').
xml/mixedElement(13, 17, ' book.').
book(1, 18).
year(18, 19, '2002').
author(18, 21, 'Buneman').
title(18, 23, 'XML in Scotland').
review(18, 25).
em(25, 26).
xml/mixedElement(26, 27, 'The ').
em(26, 28, 'best').
xml/mixedElement(26, 30, 'ever!').
```

Figure 7: Prolog translation by Lima (2012)

Take for instance the fact *author('Abiteboul', [1, 1, 1], 3)*. The first argument, *'Abiteboul'*, is the content of the XML element *author* as we can see in Figure 5. The second argument is the node number. In the **nodenumber (**the list [*1, 1, 1])*, the first index is an identifier for different *author* elements of a same parent. Note that the authors *Buneman* and *Suciu* have different values for it. The second one identifies author elements from different parents. We can see the difference between facts where the first argument is '*Buneman'*. In

this case, the second index of **nodenumbers** is different. Finally, the last index identifies the parent of author's parent, in this case, the root element. For all *author* facts in Figure 6, the third index of **nodenumber** is equal. In addition, the third argument (number 3), is used to identify weakly distinct element nodes and it is named **typenumber**. There is an example of this in Figure 5 and Figure 6. The elements *em*, which are children of element *review*, have different values for **typenumber**. Both elements are children of element *review*, but the two elements *review* are organized in a different way. This numbering makes it possible to distinguish the facts, thus avoiding confusing them.

2.3 RUNNING XML QUERIES IN PROLOG

Almedros-Jiménez *et al*. (2008) also created a way to convert an XPath to Prolog terms named "Program Specialization for XPath Expressions". Although this last work does not provide a translation from XQuery to Prolog facts and rules aiming to use Prolog engines, another one (ALMENDROS-JIMÉNEZ; BECERRA-TERÓN; ENCISO-BANOS, 2007), by the same authors, investigates how to integrate the XQuery language and logic programming. Thus, Almendros-Jiménzes *et al*. (2007) built an entire workflow of execution for executing XQuery queries on Prolog engines that includes the translation of the XML document into Prolog facts, and also the translation of the queries into Prolog rules. However, these work do not present an heuristic to help users decide the better approach to be used and there is no mention of a performance evaluation.

The work of Santos *et al.* (2012) is a first attempt to create an entire workflow of execution for running XPath over a Prolog engine and receiving an XML subtree as answer using the XMLInference approach (LIMA *et al.*, 2012a, b). In this work, the authors show a method to convert XPath queries into Prolog terms and also a method to convert the prolog query result back to XML. However, the authors do not evaluate the performance of the resulting queries.

We can note that our work stimulates the idea of an entire workflow of execution that would be completely transparent to the user. The user would continue to use XQuery as query language and besides, if needed, she could take advantage of the inference feature. This would be transparent to users because, as we will present in the next chapters, the execution times for Prolog engines is almost always better than those for native tools.

CHAPTER 3 – EXPERIMENTAL SETUP

3.1 OVERVIEW

In order to thoroughly evaluate the approach discussed in this work, we used the XBench benchmark (YAO; OZSU; KHANDELWAL, 2004). XBench provides four different types of benchmark datasets, together with a set of corresponding XQuery queries. In our work, we used three of these datasets: data centric single document (DC/SD), text centric single document (TC/SD) and text centric multiple document (TC/MD), all with about 10MB. In a single document database, the whole dataset is stored in a single file, while the multiple document database has more than one file composing the dataset. Both multiple document and single document can be data centric or text centric. In the former, the majority of the information is data, while in the later it is text. We use three of these datasets and their queries to find out heuristics that are able to indicate when it is more appropriate to use the XMLInference approach. Details of these datasets can be found in Appendix I and II.

The three selected datasets toghether provide us 52 queries. These queries are distributed between datasets as follows: 16 queries for the data centric single document (DC/SD), 17 queries for the text centric single document (TC/SD) and 19 queries for the text centric multiple document (TC/MD) dataset.

The remaining of this chapter is organized as follows. Section 3.2 describes the steps needed to start the evaluation. In this section, we show an example of query translation from XQuery to Prolog and scripts used to help us to get results in an organized and optimized way. In Section 3.3, we explain how we translated some XQueries to Prolog. Section 3.4 describes the query mechanisms used in our experiment and finally, Section 3.5 shows some environment information.

3.2 EXECUTION SCRIPTS

Before starting the execution of the queries, we need to prepare the input and the environment to execute the experiments. For both native XQuery and Prolog-based query engines, we used the application distributed by XBench to generate the XML files. With this application, we can generate the datasets mentioned earlier in this chapter. Notice that this was done a single time during the setup of the experiments.

After running the XBench application, we were able to start the execution flow for native XQuery mechanisms. However, this is not true for the Prolog engines. For them, there are some additional steps.

As already mentioned, we used the XMLInference prototype to convert XML files into Prolog rules and facts. To convert XML elements into Prolog facts and rules we just need to run the XMLInference prototype with the correct parameters. Details about XMLInference were discussed in Chapter 2. Besides, we have to create Prolog queries that correspond to the XQuery queries of the benchmark, and this requires a significant effort.

```
for $word in doc('doc.xml')/dictionary/e
where some $item in $word/ss/s/qp/q
    satisfies $item/qd eq "1900"
return
    $word
```

```
pConsulta06_01([]).
pConsulta06_01([H|T]) :- pE(H), pConsulta06_01(T).
pConsulta06:-setof(ID,A^B^C^D^E^(qd(B,A,'1900'), q(C,B), qp(D,C), s(E,D),
    ss(ID,E)), LISTA), pConsulta06_01(LISTA).
```

Figure 8: Comparing XQuery with a Prolog query

As a simple example, Figure 8 shows an XQuery (top) and a Prolog query (bottom). This query returns words in a dictionary entry that were quoted in a certain year (1900). As we can see, the translation to Prolog is not trivial. This translation step was the hardest task in our methodology, and where we spent most of the time.

Although we could start the execution flow at this point, it would be very hard and susceptible to errors. Running each query eleven times in a non-automated way is not a good idea. Because of this, we created some shell scripts to do this job.

```
#!/bin/bash

ls -1 /workPath/XQueryQueries/*.xquery | while read file
do
   output=`basename $file`
   for i in 0 1 2 3 4 5 6 7 8 9 10
   do
      echo "****** Executing XQuery inside file ${file}! ******"
      /usr/bin/time -f %E /opt/sedna/bin/se_term -file ${file} base_name
2>> resultados.${output}.txt > /dev/null
      echo "******** Fim da XQuery ********\n\n"
   done
done
```

Figure 9: Shell Script to call Sedna and get time results

Figure 9 shows our shell script that calls Sedna eleven times for a given query and gets the execution time at each run. The execution time is retrieved by Linux through the *time* command. Note that the query result got from Sedna is sent to Linux's null device. Thus, we avoid spending time displaying the results on the screen or writing them to a file. To compute the mean and standard deviation, the Script in Figure 10 uses the output generated by the script in Figure 9 to generate a second file with this information.

```
#!/bin/bash

ls -1 result* | while read file
do
   i=0
   while read line
   do
      number=`echo $line | cut -c4-`
      vetor[i]=$number
      i=`expr $i + 1`
   done < $file
   # Computing the average
   i=0
   soma=0
   while [ $i -lt ${#vetor[@]} ]
   do
      soma=`echo "scale=7; ($soma + ${vetor[i]})" | bc`
      i=`expr $i + 1`
   done
   avg=`echo "scale=7; (${soma}) / 10 " | bc`
   # Computing the standard deviation
   i=0
   soma=0
   while [ $i -lt ${#vetor[@]} ]
   do
      aux=`echo "scale=7; (${vetor[i]} - $avg) ^ 2" | bc`
      soma=`echo "scale=7; ($soma + $aux)" | bc`
      i=`expr $i + 1`
   done
   dv=`echo "scale=7; sqrt($soma / (${#vetor[@]} - 1))" | bc`
   #imprimindo valores
   query=`echo $file | awk -F "." '{print $2$3}'`
   echo "The $query has average $avg and standard deviation $dv"
done
exit 0
```

Figure 10: Shell script to compute Standard deviation and Mean

After generating the XML dataset, post-processing it to generate Prolog facts and rules, translating the benchmark queries into Prolog, and building scripts to automate the execution, we are able to start the execution flow of our experiments.

3.3 TRANSLATING XML QUERIES TO PROLOG QUERIES

In this section, we give the rationale we used to translate XML queries into Prolog queries for our experiments. We show how we translate the most common XQuery operators. For all translated queries, we omit the functions responsible for printing the XML tags in the query results. This makes the queries cleaner and easier to understand.

Figure 11 shows the schema of the Text Centric Multiple Document (TC/MD) dataset provided by XBench. The example queries we use in this section are run over this schema.

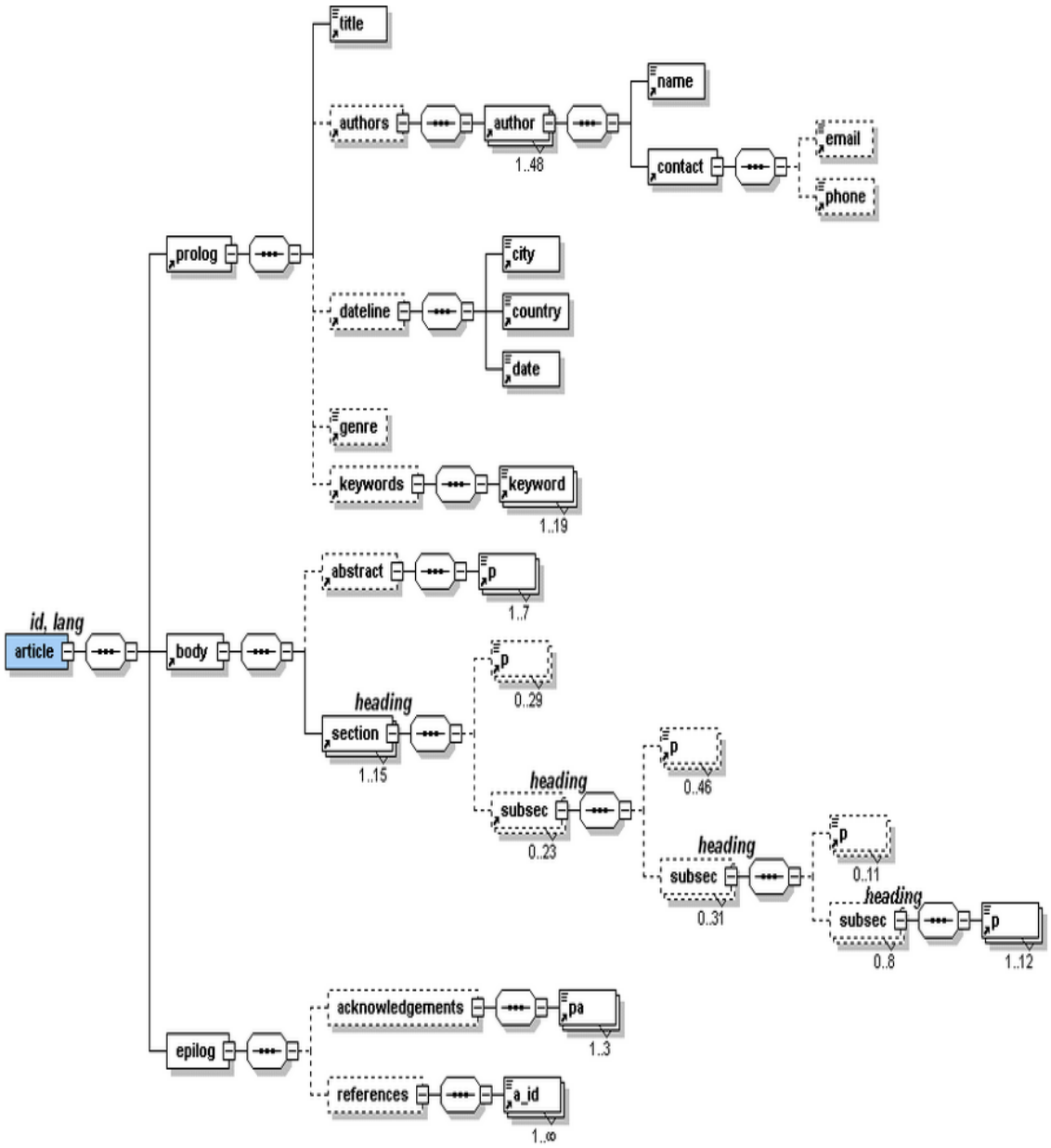


Figure 11: XBench TC/MD diagram

```
for $art in input()/article[@id="1"]
return
   $art/prolog/title
```

Figure 12: Example of XQuery from XBench

Figure 12 shows a simple XQuery that has a *for* clause with a filter that queries for a specific id, and a *return* clause. For queries like this, we can use the key information **[@id="1"]** to simplify the translated query (notice that the key, in this case, is a domain knowledge that would need to be provided to the query translation process). Inside the base of Prolog facts, there must exist only one fact *id* with value "1". Because of this, the translated Prolog query can go directly to this fact. We can see the translated query in Figure 13. To connect the facts, we use the variables _1 and _2. Finally, the value of prolog title will be assigned to variable T. These two variables contain the value of the fact's id. We use this way of connecting the facts in all translated Prolog queries.

```
queryWithID :- id(_1,_,'1'), prolog(_1,_2), title(_2,_,T).
```

Figure 13: Translated Prolog query

The query shown in Figure 14 there is a *where* clause instead of a key filter. Therefore, we can't access a single fact directly. Instead of this, we must search the whole base of facts to match the results. Figure 15 shows how we can get this done in Prolog. To search for all facts with predicate *genre* and value "Machine Learning", we use the built in Prolog function named *findall*. This function returns a list of ids into the variable LIST. With these ids, we can retrieve all related facts. The Prolog rule *queryWithWhere_01* iterates through a list to build the result.

```
for $art in input()/article/prolog
where $art/genre = "Machine Learning"
return
   $art/title
```

Figure 14: XQuery example using *where*

```
queryWithWhere :- findall(X,genre(X,_,'Machine Learning'),LIST),
queryWithWhere_01(LIST).

queryWithWhere_01([]).

queryWithWhere_01([H|T]) :- title(H,_,TITLE), queryWithWhere_01(T).
```

Figure 15: Translated Prolog query from XQuery in Figure 14

We used *findall* to search for all genre predicates with last parameter equals “Machine Learning”. But if we desire to search for the different values for *genre* predicate, what do we have to use? In this case, we should use *setof* function instead of *findall* function.

Another important and very used opetator in XQuery is *order by*. Adding the order by operator to query shown in Figure 14, we generate the query in Figure 16. There are two important things that need to be used when converting an XQuery query with *order by* operator into a Prolog query. First, we need to use “-” to separate two variables in the *findall* command, where the left variable is used to sort the list and the right one is the content to be sorted. In Figure 17, we first use the built-in Prolog function *findall* to get a list of article titles (using variable *T*) plus genre’s parent id (using variable *X*), where their genre is "Machine Learning". Each tuple article title and genre’s parent id is separated by an hyphen (*T-X*). The next step is to sort this list. To sort this list, we use the built-in Prolog function *sort*. This function receives a list and returns an ordered list. When the list’s item is a hyphen-separated value, the function *sort* uses the left part of value in the sorting procedure. Finally, the ordered list is passed to the *queryWithOrderBy_01* function. This function is responsible to iterate through the list and retrieve the values of dateline country. The function uses the genre’s parent id to access the predicates of dateline and country.

```
for $art in input()/article/prolog
where $art/genre = "Machine Learning"
order by $art/title
return
   $art/dateline/country
```

Figure 16: XQuery example using *order by*

The techniques discussed in this section do not cover all techniques used in our experiments, but they can be used as an initial guide to translate XQuery queries to Prolog queries. All translated queries used in our experiments are available in Appendix II.

```
queryWithOrderBy :- findall(T-X,(genre(X,_,'Machine Learning'),
   title(X,_,T)),LIST), sort(LIST,ORDERED), queryWithOrderBy_01(ORDERED).

queryWithOrderBy_01([]).

queryWithOrderBy_01([H-N|T]) :- dateline(N,_,DATELINE),
   country(DATELINE,_,COUNTRY), queryWithOrderBy_01(T).
```

Figure 17: Translated Prolog query from XQuery in Figure 16

3.4 QUERY MECHANISMS

In this section, we describe each tool used in our tests. The results generated by them were used to build our heuristic.

Among many Prolog engines, we chose two approaches: SWI-Prolog and YAP. We were looking for approaches that give us an API to integrate with XMLInference prototype in the future.

YAP (COSTA; ROCHA; DAMAS, 2012) is a high-performance Prolog compiler developed by the Research Center for Advanced Computing Systems, University of Porto. Its engine is designed with a number of optimizations in order to obtain high performance. It was written in C and can be compiled on 32 and 64 bits platforms.

SWI-Prolog (JAN WIELEMAKER *et al.*, 2012) is an open source implementation for the logic programming language Prolog. Besides providing a Prolog environment where you can upload and execute files through its interactive command interface, you can also use the same through their libraries available to other programming languages.

We also needed native XML query processors so that we could compare their performance with the performance of the Prolog query engines. Sedna and eXist are very popular in the academic environment. Besides, they are stable and fast tools to query and to manage XML databases. We also chose a command line XQuery processor. The most stable one is Galax. Based on this information, we chose these three tools to compare with Prolog engines.

Galax (SIMÉON *et al.*, 2000) is an XQuery query processor that runs queries directly on XML documents that are stored in the file system of the operating system. The processor is an open source implementation of XQuery and one of the first XQuery processors implementations to appear. Since it runs over files stored in the file system, no indexing is provided.

Sedna (FOMICHEV; GRINEV; KUZNETSOV, 2006) is a native XML database management system. This system was designed so that its modules presented good performance on query processing. Sedna also provides Java libraries for applications that can communicate with the database system.

The eXist native XML database project (MEIER, 2003) is strongly based in open standards and open source. It provides an integrated environment where the user can manage databases and make quick queries in some documents. In addition to a GUI interface, the user can use the Java API to add and retrieve information from XML databases.

3.5 SETTING ENVIRONMENT

The experiments were performed on an Intel Dual Core 2.16 GHz machine, with 2.0 GB of RAM memory and 32-bit Linux (Ubuntu) operating system. When the experiments were run, we tried to avoid interferences of other applications. In other words, we did not start other user applications when the experiments were being executed.

In our experiments, we focused on the evaluation of execution times for each evaluated tool and compared them. At this moment, we are not interested in the consumption of other resources such as memory and disk, because for our objectives, response time is the most important requirement.

Aiming to verify that the results gathered would be independent of the query engine, we chose some different approaches. In particular, we used the XML query engines Galax, Sedna and eXist. YAP and SWI-Prolog were used as the Prolog engines coupled to the XMLInference approach.

Each query was written in the underlying language of the query engine for each evaluated tool. We had to adapt the way to obtain the execution time in each of these approaches, because of the heterogeneity between them. For example, to get the total execution time of a query running on Galax at Linux, you just need to use the time Linux/Unix command, but this is not the same for SWI-Prolog. In SWI-Prolog, we used the predicate *time(:Goal)* where Goal is the predicate for which we desire to get the execution time. Each query was run 11 times on each tool and the first run was discarded. We discarded the first one to avoid getting an execution time during the process bootstrap or while the process are loading something into memory. The mean and standard deviation were computed on the remaining executions. Figure 18 shows this execution flow.

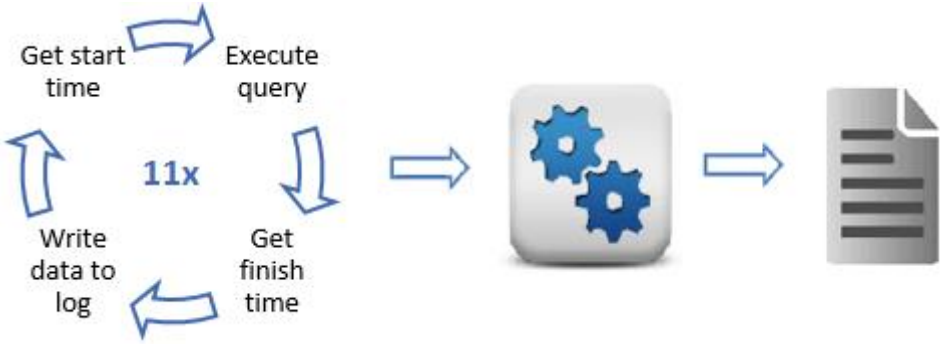


Figure 18: Execution flow

CHAPTER 4 – AN HEURISTIC FOR XML QUERY PROCESSING

4.1 OVERVIEW

In this chapter, we show the results achieved for each approach and discuss them aiming to build an heuristic. It is organized as follows. In Section 4.2, we present the results gathered by each engine in the evaluated datasets, and discuss them. Finally, in Section 4.3, we provide the heuristics, derived from the results gathered by the query mechanisms, that can be a guide for one deciding when it is better to use Prolog to run XQuery, and when it is not.

4.2 RESULTS

After executing all query mechanisms for all datasets, we collected the results and organize them in Table 1, Table 2 and Table 3.

Table 1: Query Mechanisms Execution time for DC/SD (in seconds)

Query	Galax	Sedna	eXist	Yap	SWI-Prolog
Q01	5.0600	0.060	0.0565	0.1835	0.0391
Q02	5.0400	0.064	0.1184	0.0191	0.0035
Q03	7.0900	0.258	1.2504	0.0132	1.2307
Q04	4.9200	0.057	0.0518	0.0011	0.0022
Q05	4.9400	0.056	0.0564	0.1168	0.0253
Q06	5.2500	0.121	0.4472	0.2162	0.1099
Q07	5.1800	0.084	0.1926	0.2016	0.0689
Q08	4.9700	0.055	0.2018	0.0845	0.0170
Q09	5.0200	0.055	0.0522	0.0044	0.0011
Q10	5.4800	0.121	0.0760	0.0044	0.0335
Q11	5.0400	0.067	0.0749	0.0066	0.0077
Q12	4.9500	0.055	0.0566	0.0761	0.0140
Q14	4.9900	0.055	0.0759	0.0172	0.0078
Q17	6.2200	0.130	0.1319	0.2834	0.0341
Q19	5.3300	0.068	0.2202	0.0134	0.0033
Q20	6.3000	0.076	0.1513	0.0154	0.0892

Table 2: Query Mechanisms Execution time for TC/SD (in seconds)

Query	Galax	Sedna	eXist	Yap	SWI-Prolog
Q01	5.7700	0.046	0.0249	0.2821	0.1500
Q02	6.2100	0.118	0.4486	0.0979	0.0553
Q03	6.7600	2.647	6.3338	0.0932	0.1322
Q04	5.8000	0.119	0.1254	0.0084	0.0011
Q05	5.5800	0.044	0.0256	0.2764	0.0695
Q06	6.6300	0.154	0.6823	0.4650	1.4655
Q07	5.6900	0.055	0.0793	0.1055	0.0715
Q08	5.6100	0.044	0.2028	0.1368	0.0451
Q09	10.4500	0.044	0.1847	0.1368	0.0448
Q10	*	*	*	*	*
Q11	5.7600	0.045	0.0265	0.1068	0.0180
Q12	5.6000	0.055	0.0207	0.2782	0.1202
Q13	5.7600	0.044	0.0234	0.0066	0.0012
Q14	5.6600	0.055	0.0276	0.0011	0.0011
Q17	6.6900	0.391	3.2724	0.8726	5.3123
Q18	6.6600	0.563	2.0901	1.8609	18.5170
Q19	5.7800	0.044	0.0518	0.9890	0.0019

** Query 10 was discarded because its textual description does not match its XQuery implementation in XBench.*

We couldn't execute the queries in Galax for the text centric multiple document dataset because Galax doesn't deal with collections. Without this feature, we cannot use the *collection* function of XQuery, and so the queries would not run successfully. There is a solution for this problem in (RODRIGUES; BRAGANHOLO; MATTOSO, 2011), but as Galax got the worst results for both data centric and text centric single document dataset, we decided not to evaluate Galax in this work.

To analyze these results, we first compare the Prolog engines to decide which we are going to choose. After, we compare the results of the native XML databases. Finally, we compare both winner approaches. All execution times were normalized to make the

differences in runtime easier to perceive. The normalization method consists in getting the higher execution time for all approaches in a specific query and call it the max value. The remaining execution times for that same query are then divided by this max value. Thus, the values in the graphics are always between 0 and 1, where 1 is the max value. Because of high processing times, we do not include Galax in our comparison. Galax does its search using the file system with no indexing. We believe this is the main reason for the high processing times.

Table 3: Query Mechanisms Execution time for TC/MD (in seconds)

Query	Sedna	eXist	Yap	SWI-Prolog
Q01	0.0850	0.0162	0.000	0.0000
Q02	0.0440	0.0136	0.000	0.0000
Q03	0.0570	0.0682	0.000	0.0066
Q04	0.0440	0.0084	0.0011	0.0007
Q05	0.0440	0.0091	0.0011	0.0007
Q06	0.0440	0.0278	0.0268	0.0653
Q07	0.0440	0.0428	0.0275	0.0580
Q08	0.0450	0.2732	0.0077	0.0007
Q09	0.0440	0.3269	0.0077	0.0007
Q10	0.0460	0.0259	0.0052	0.0007
Q11	0.0420	0.0165	0.0000	0.0007
Q12	0.0760	0.0078	0.0843	0.0738
Q13	0.0440	0.0124	0.0155	0.0007
Q14	0.0440	0.0104	0.0000	0.0007
Q15	0.0880	0.1286	0.0043	0.0101
Q16	0.0600	0.0134	0.0849	0.0718
Q17	0.0460	0.1752	0.0324	0.0880
Q18	0.0610	0.0478	0.0323	0.0972
Q19	0.0460	0.1348	0.0058	0.0007

4.2.1 PROLOG ENGINES

Figure 19 (DC/SD), Figure 20 (TC/SD) and Figure 21 (TC/MD) compare the performance of the Prolog engines. In both graphics, the X-axis is the query number and the Y-axis is the normalized execution time.

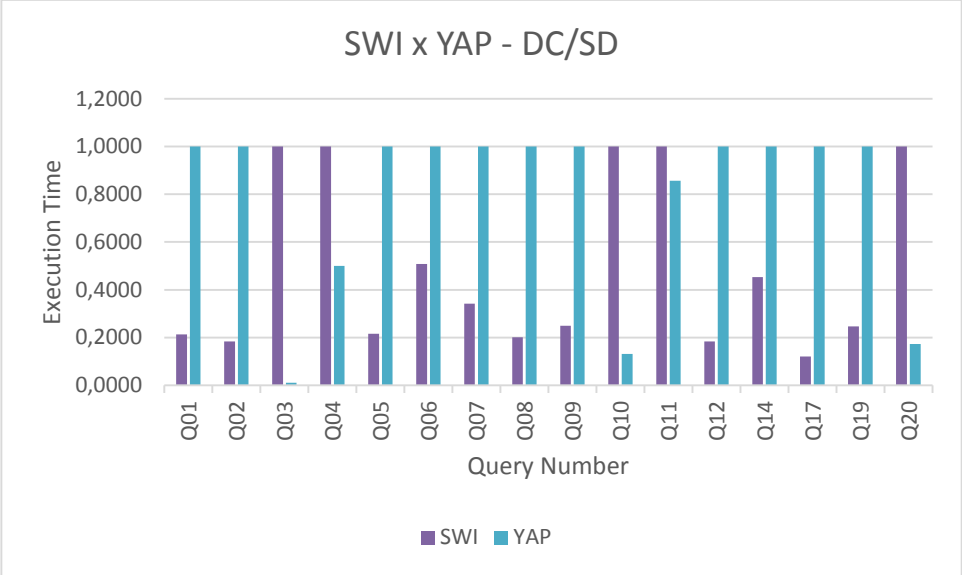


Figure 19: Comparing SWI versus YAP execution times – DC/SD

Analyzing Figure 19 (DC/SD), we observe that SWI was better than YAP in 11 of 16 queries. The five queries where SWI was worst than YAP use relational operators (>, <, >=, <= or =), *order by*, *distinct* or the *position*() XPath Function. Queries Q10 and Q11 have an *order by* operator and relational operators. Query Q3 have a relational operator and a *distinct* operator. Query Q4 uses the *position*() XPath function (CLARK; DEROSE, 1999). Finally, Q20 has also a relational operator.

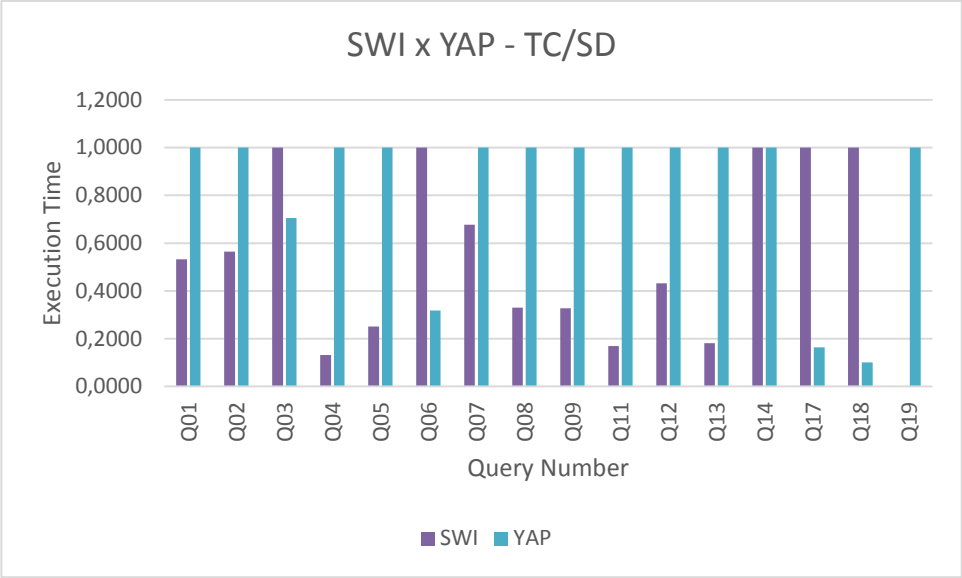


Figure 20: Comparing SWI versus YAP execution times – TC/SD

In Figure 20 (TC/SD), we can see that SWI was better than YAP in 11 of 16 queries. There was a tie in query Q14. In the four queries where SWI was worst than YAP, we can observe the following. Queries Q17 and Q18 use the XPath *contains()* function to execute text search in database. Query Q3 has a *distinct* operator. Finally, Q6 used the XQuery *some* operator.

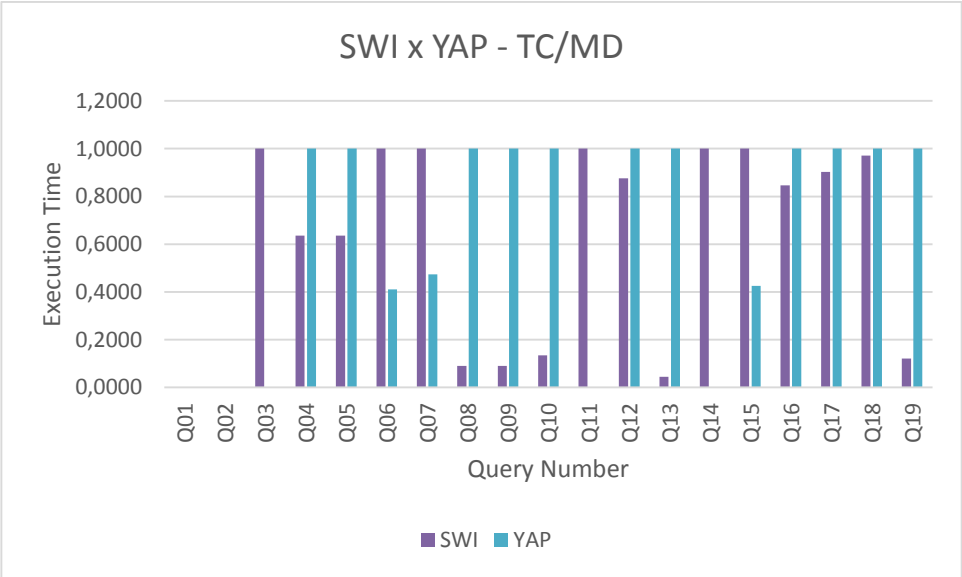


Figure 21: Comparing SWI versus YAP execution times – TC/MD

In Figure 21 (TC/MD), we see that SWI was better than YAP in 11 of 19 queries. For queries Q1 and Q2 the SWI and YAP got the same result. For the six queries where SWI was

worst than YAP, we can observe some points related to XQuery. Query Q3 has a *distinct* operator. Queries Q6 and Q7 use the XPath *contains()* function to execute text search in the database. Query Q11 have an *order by* operator. Finally, the queries Q14 and Q15 use an XQuery *empty* function.

The implementation made in Prolog to all of these highlighted points seems to work best when we use YAP. However, in most cases, SWI-Prolog had the best results. Because of this fact, we chose SWI as our main Prolog approach to compare with the native XML engine.

4.2.2 NATIVE XML TOOLS

In Figure 22 (DC/SD), we see that Sedna was better than eXist in 12 of 16 queries. For the queries where Sedna was worst than eXist in this dataset, we can observe some points related to XQuery. Query Q1 is a simple query that uses a key. However, this query asks for a large amount of data. Query Q4 has the XPath *position()* operator. Query Q9 is a search using a key. Finally, query Q10 has two points. It uses an *order by* operator and its clause where defines a range between dates.

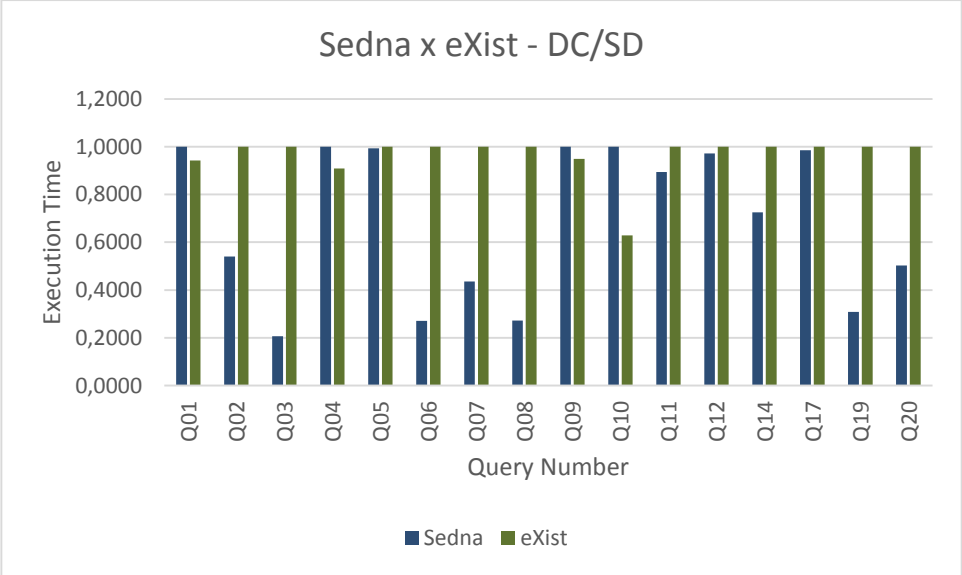


Figure 22: Comparing Sedna versus eXist execution times – DC/SD

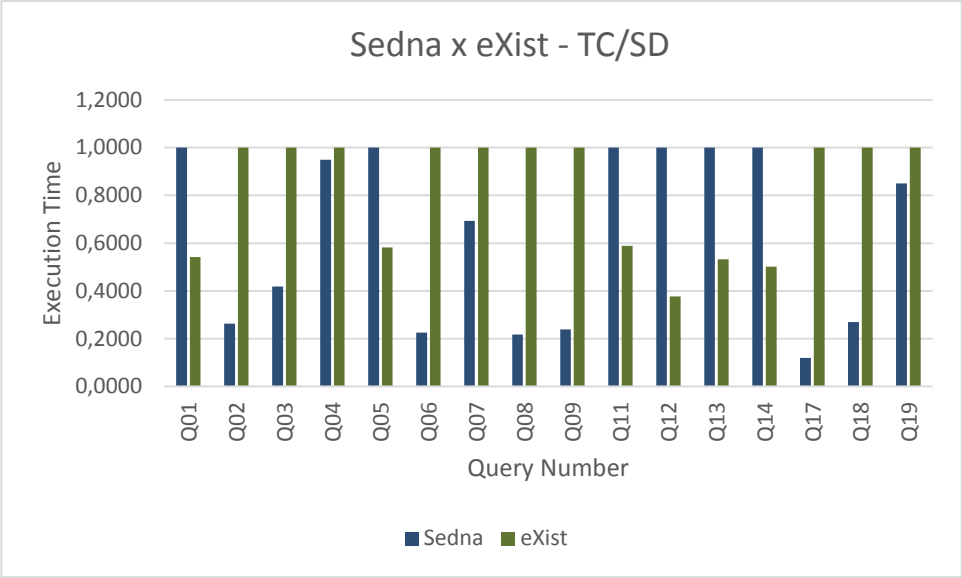


Figure 23: Comparing Sedna versus eXist execution times – TC/SD

In Figure 23 (TC/SD), we see that Sedna was better than eXist in 10 of 16 queries. For the queries where Sedna was worst than eXist in this dataset, we can observe some points related to XQuery. Query Q1 is a simple query that searches in the entire dataset (full scan). However, this query asks for a large amount of data. Queries Q5, Q12 and Q13 are very similar to Q1, but they ask for a fewer amount of data. Finally, query Q11 has an *order by* operator.

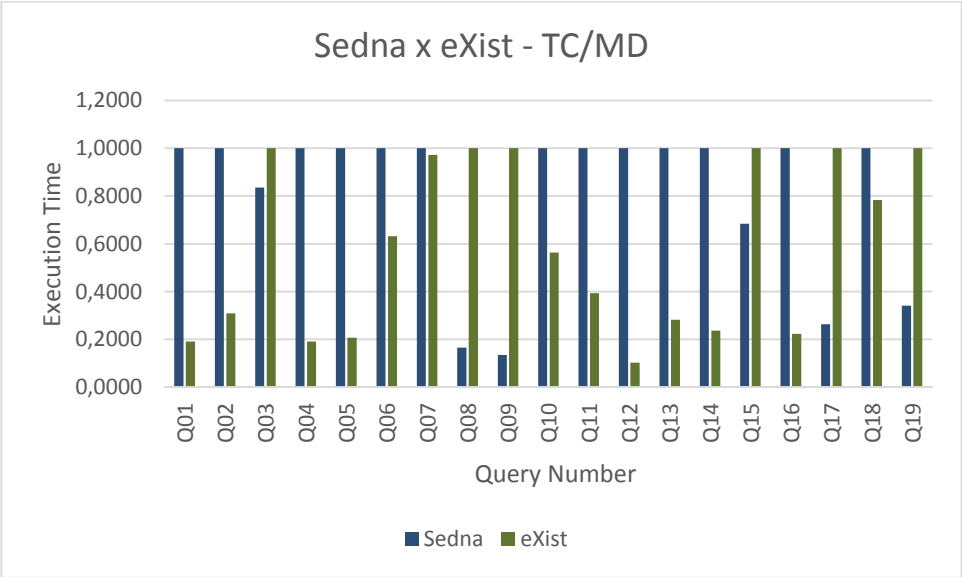


Figure 24: Comparing Sedna versus eXist execution times – TC/MD

Unlike the two previous datasets, in Figure 24, we see that eXist was better than Sedna in 13 of 19 queries. For the queries where eXist was worst than Sedna in this dataset, we can observe some points related to XQuery. Query Q3 has a *distinct* operator. Queries Q8 and Q9 use a key in their searches. Query Q15 uses the *empty* function. Query Q17 does a search into text using the XPath *contains()* function. Finally, query Q19 is a little more complex. It uses a key to fetch a value that is used to match with others in the entire document.

Looking at the graphics in Figure 22 and Figure 23, both about single document datasets, we see that Sedna gets the best results for almost every query. However, when we look at the graphic in Figure 24, we see that eXist gets the best results for almost every query. Because of this, we decided to compare Sedna with SWI for single document datasets and eXist with SWI for multiple document dataset.

4.2.3 XML NATIVE TOOLS VERSUS SWI

Looking at the graphs in Figure 25, Figure 26 and Figure 27, we see that the SWI-Prolog engine got good execution times when compared with the other XML approaches. Not even Sedna and eXist, that are native XML data base systems, got times as good as SWI-Prolog engine. Since in this section we analyze the approaches which gathered the best individual results, we also present the statistical significance for each query evaluated between them.

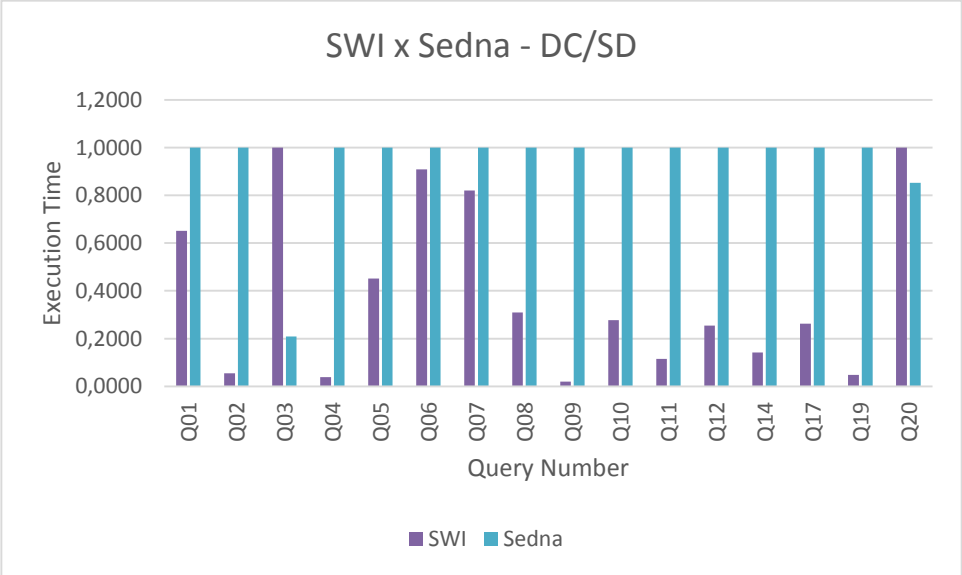


Figure 25: Comparing Sedna versus SWI execution times – DC/SD

Table 4: Statistical significance for Sedna versus SWI execution times – DC/SD

Queries	p-value
Q01	0.0001
Q02	0.0001
Q03	0.0001
Q04	0.0001
Q05	0.0001
Q06	0.0001
Q07	0.0001
Q08	0.0001
Q09	0.0001
Q10	0.0001
Q11	0.0001
Q12	0.0001
Q14	0.0001
Q17	0.0001
Q19	0.0001
Q20	0.0001

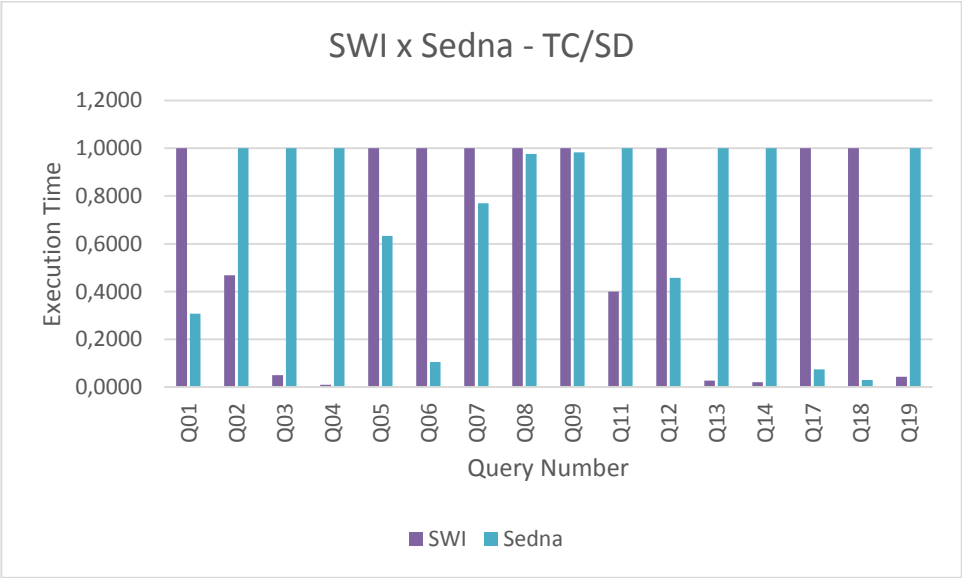


Figure 26: Comparing Sedna versus SWI execution times – TC/SD

Table 5: Statistical significance for Sedna versus SWI execution times – TC/SD

Queries	p-value
Q01	0.0001
Q02	0.0001
Q03	0.0001
Q04	0.0001
Q05	0.0001
Q06	0.0001
Q07	0.0001
Q08	0.0001
Q09	0.0042
Q11	0.0001
Q12	0.0001
Q13	0.0001
Q14	0.0001
Q17	0.0001
Q18	0.0001
Q19	0.0001

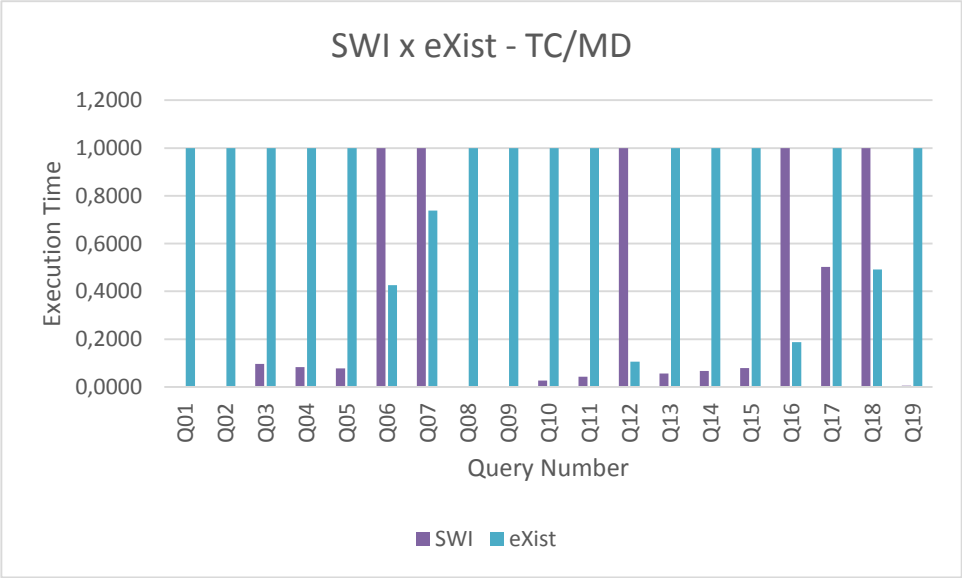


Figure 27: Comparing SWI versus eXist execution times – TC/MD

Table 6: Statistical significance for Sedna versus SWI execution times – TC/MD

Queries	p-value
Q01	**0.0001**
Q02	**0.0001**
Q03	**0.0001**
Q04	**0.0004**
Q05	**0.0001**
Q06	**0.0001**
Q07	**0.0001**
Q08	**0.0001**
Q09	**0.0042**
Q10	**0.0013**
Q11	**0.0001**
Q12	**0.0001**
Q13	**0.0001**
Q14	**0.0001**
Q15	**0.0001**
Q16	**0.0001**
Q17	**0.0001**
Q18	**0.1172**
Q19	**0.0001**

Analyzing Figure 25, we see that SWI was better than Sedna in 14 of 16 queries for data centric single document dataset. When we compare SWI with Sedna for text centric single document dataset, Figure 26, we observe that SWI won Sedna in 7 of 16 queries, but queries Q08 and Q09 were almost a tie. In Figure 27, for text centric multiple document dataset, we note that SWI got the best results in 12 of 19 queries.

To calculate the statistical significance we used Student's *t*-distribution. The input for the distribution method are the 10 execution times for both query mechanisms (SWI and Sedna). Our results are statistically significant for all queries, except for Q18 in Table 6. The p-value **0.1172** means that the result for the query is considered not to be statistically significant.

4.2.4 COMPARING ALL APPROACHES

Now, we look at the two Prolog engines and the XML native tools together in Figure 28 (DC/SD), Figure 29 (TC/SD) and Figure 30 (TC/MD).

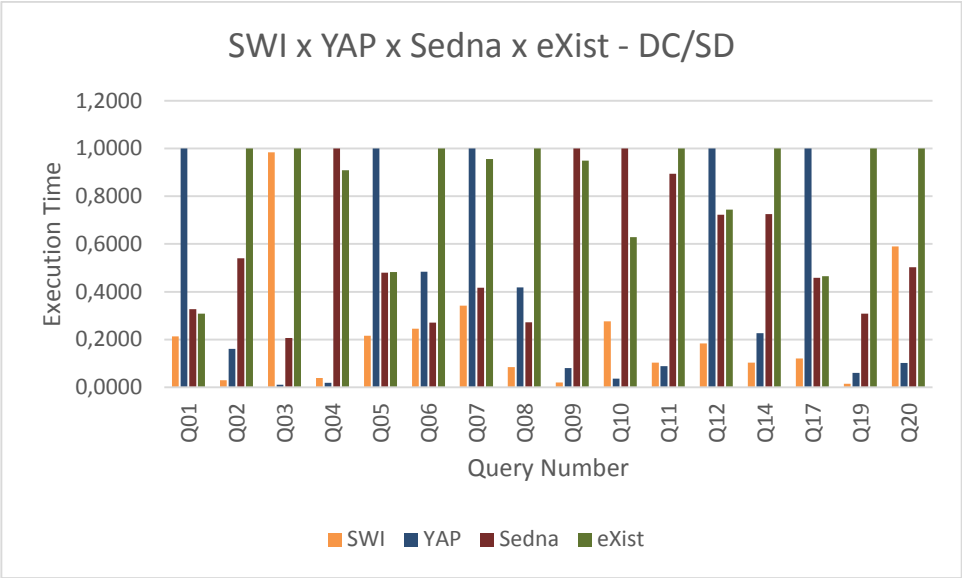


Figure 28: Comparing SWI, YAP, Sedna and eXist execution times – DC/SD

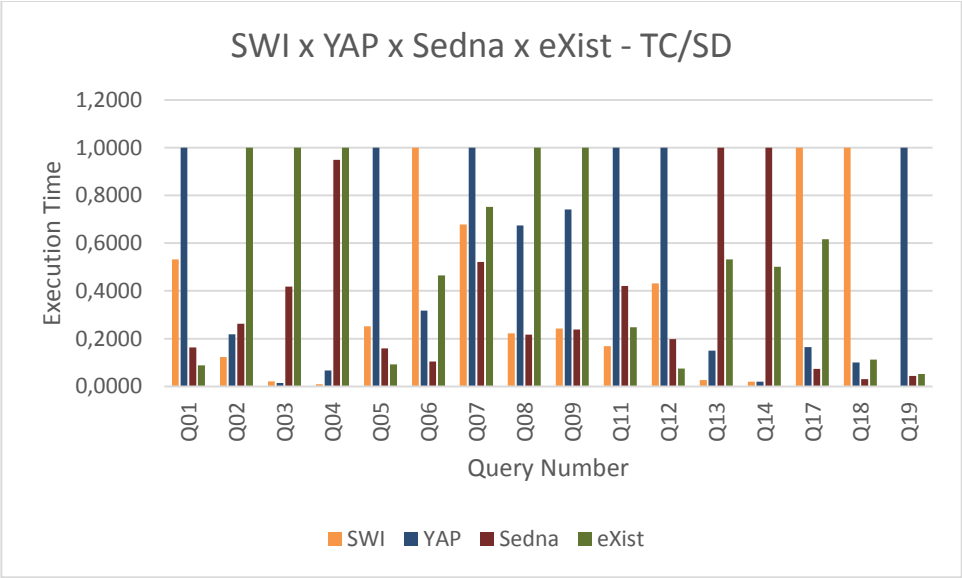


Figure 29: Comparing SWI, YAP, Sedna and eXist execution times – TC/SD

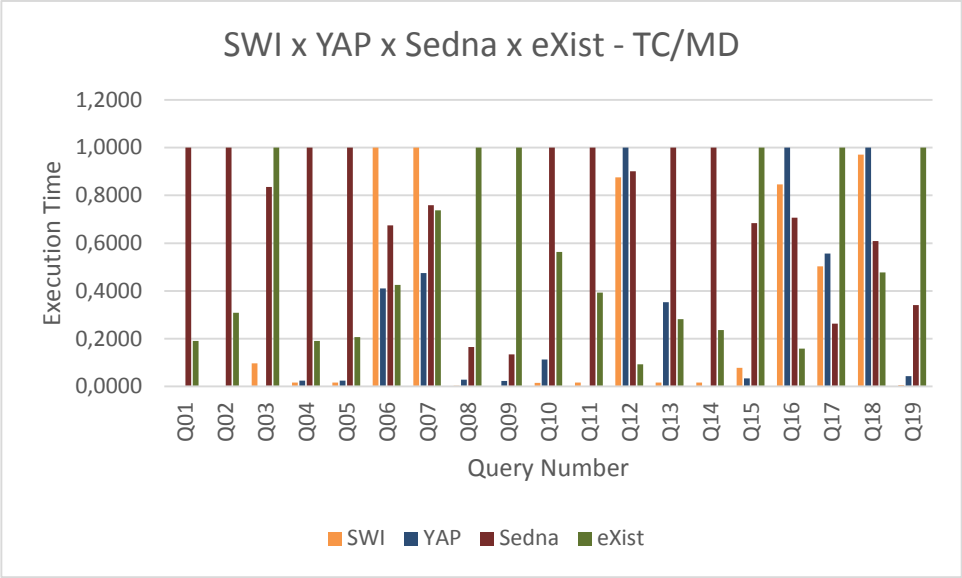


Figure 30: Comparing SWI, YAP and Sedna execution times – TC/MD

When we compare all the results together, we notice that sometimes the YAP Prolog engine beats the native XML engines, while SWI does not. Take, for instance, query Q20 in Figure 28 and queries Q6 and Q7 of Figure 30. Although SWI was chosen to be compared with the native XML approaches, in these queries YAP beats SWI and also Sedna and eXist. This fact may justify a hybrid approach because both SWI and YAP are open source.

We present a deeper discussion of the results in the next section.

4.3 DISCUSSION

In this subsection, we discuss our experimental results. We selected some technical features to analyze for each query. They are: type of search (A), sorting triggered by the schema (B), text search (C), sorting triggered by query (D), quantifier operator (E), filter operator (F), large amount of related data (G) and distinct values (H). These technical features are explained next.

(A) Type of search. There are two kinds of search: simple search and full search. Simple search occurs when we are querying by unique elements like the primary key in the relational context or using the position operator of XPath (for instance, */catalog/item[1]/title*). We have to explain two points. First, in XMLInference approach, simple search occurs only when the query searches for the first occurrence of an element. Queries that search for positions different than one require a *findall* in the translated query and because of this, all

content of file must be read. Second, in XML files, even when the file does not have an explicit key, if we know the domain, we can assume an element as a key.

On the other hand, full search occurs when the whole file has to be traversed to answer the query. In other words, after getting the first matching element, the search must continue to check if there exist additional matching elements.

(B) Sorting triggered by schema. As explained in previous sections, XMLInference approach converts an XML dataset to Prolog facts and rules and this fact generates a problem about elements sorting. When an XML Schema or DTD defines a choice element as the element *vfl* presented in Figure 31, the sub elements *vd* e *vf* can appear in a mixed way. When we convert the XML file to Prolog facts and rules, we sometimes lose this information. Specifically, we loose this information everytime the subelements are leaf nodes. Because of this, we changed the original XMLInference approach by adding an additional element/attribute ID to this type of element. This makes it possible to know what the original sequence of sub elements was. This feature solves the sorting problem, but when a sorting is triggered by schema, the execution time rises.

(C) Text Search: Text search always occurs when the XQuery has the *contains()* XPath function. The *contains()* function searches for substrings.

(D) Sorting triggered by query: Sorting triggered by query occurs when XQuery has the *order by* operator.

(E) Quantifier operator: XQuery has operators to execute the universal and existencial quantifiers. Thus, every time a query has an XQuery operator *every* (universal quantifier) or *some* (existencial quantifier), we classify it as a query with quantifier operator.

(F) Filter operator: A filter operator occurs when a XQuery has in its *where* clause comparisons made with operators *equal (=), not equal (!=), greater than (>), greater than or equal to (>=), lower than (<)* and *lower than or equal to (<=).*

(G) Large amount of related data: The technical feature amount of related data occurs when the return statement asks for large amounts of data or when a large quantity of data must be analyzed to find the answers.

(H) Distinct values: XQuery allows queries to remove duplicates in the query answer. This is done by the *distinct* XML operator.

```
<xs:element name="vfl">
      <xs:complexType>
            <xs:choice maxOccurs="45">
                  <xs:element ref="vd" minOccurs="0"/>
                  <xs:element ref="vf" maxOccurs="4"/>
            </xs:choice>
      </xs:complexType>
</xs:element>
```

Figure 31: Element definition that causes a sorting triggered by schema

Table 7, Table 8 and Table 9 show us the technical features for all queries and datasets. Analyzing these tables, we build our heuristic that is presented in Table 10.

Table 7: Technical features for data centric single document

Query Number	A	B	C	D	E	F	G	H
Q01	Simple	No	No	No	No	No	No	No
Q02	Full	No	No	No	No	No	No	No
Q03	Full	No	No	No	No	Yes	No	Yes
Q04	Full	No	No	No	No	No	No	No
Q05	Simple	No	No	No	No	No	No	No
Q06	Full	No	No	No	Yes	No	No	No
Q07	Full	No	No	No	Yes	No	No	No
Q08	Simple	No	No	No	No	No	No	No
Q09	Simple	No	No	No	No	No	No	No
Q10	Full	No	No	Yes	No	Yes	No	No
Q11	Full	No	No	Yes	No	Yes	No	No
Q12	Simple	No	No	No	No	No	No	No
Q14	Full	No	No	No	No	Yes	No	No
Q17	Full	No	Yes	No	No	No	No	No
Q19	Full	No	No	No	No	No	No	No
Q20	Full	No	No	No	No	Yes	No	No

Analyzing Figure 28 and Table 7, we can deduce that for almost every query in data centric single document benchmark, SWI-Prolog is the best approach because it gets the best execution times. Exceptions for this behavior are queries Q03, Q10, Q11 and Q20. All of these queries are full search (A) and have filter operator (F). Besides, queries Q10 and Q11 have also sorting triggered by query (D). The queries Q03 and Q14 have almost the same

technical features, but for Q03, SWI-Prolog gets bad results. On the other hand, it gets good results for query Q14. The difference between them is the distinct values technical feature (H). This fact seems to be the cause of the bad result for query Q03. However, YAP has the best result for query Q03 among all approaches.

Table 8: Technical features for text centric single document

Query Number	A	B	C	D	E	F	G	H
Q01	Full	Yes	No	No	No	No	No	No
Q02	Full	No	No	No	No	No	No	No
Q03	Full	No	No	No	No	No	No	Yes
Q04	Full	No	No	No	No	No	No	No
Q05	Full	Yes	No	No	No	No	No	No
Q06	Full	Yes	No	No	Yes	No	No	No
Q07	Full	No	No	No	Yes	No	No	No
Q08	Full	No	No	No	No	No	No	No
Q09	Full	Yes	No	No	No	No	No	No
Q11	Full	No	No	Yes	No	No	No	No
Q12	Full	Yes	No	No	No	No	No	No
Q13	Full	No	No	No	No	No	No	No
Q14	Full	No	No	No	No	No	No	No
Q17	Full	No	Yes	No	No	No	Yes	No
Q18	Full	No	Yes	No	No	No	Yes	No
Q19	Full	No	No	No	No	No	No	No

In Figure 29 and Table 8, we can observe that SWI gets good execution times for most queries in TC/SD dataset, but the XML native tools also make a good job. XML native approaches wins eight times and Prolog Engines wins the same quantity of times. In some cases, we have an imperceptible difference. This fact happens in queries Q08 and Q09.

We can highlight queries Q01, Q05, Q06, Q07, Q09, Q12, Q17 and Q18 from Table 8 where XML native tools were better than Prolog engines. All queries are full search (A). For queries Q01, Q05, Q06, Q09 and Q12, additional information can be interesting. All of them have sorting triggered by schema (B). For queries Q06 and Q07, we can observe the quantifier operator (E). Queries Q17 and Q18 have also text search (C) and large amount of related data (G) technical features. Query Q11 has the sorting triggered by query (D), but this fact does not affect its performance.

Table 9: Technical features for text centric multiple document

Query Number	A	B	C	D	E	F	G	H
Q01	Simple	No	No	No	No	No	No	No
Q02	Full	No	No	No	No	No	No	No
Q03	Full	No	No	No	No	No	No	No
Q04	Full	No	No	No	No	No	No	No
Q05	Simple	No	No	No	No	No	No	No
Q06	Full	No	Yes	No	No	No	No	No
Q07	Full	No	Yes	No	No	No	No	No
Q08	Simple	No	No	No	No	No	No	No
Q09	Simple	No	No	No	No	No	No	No
Q10	Full	No	No	Yes	No	No	No	No
Q11	Full	No	No	Yes	No	No	No	No
Q12	Simple	No	No	No	No	No	Yes	No
Q13	Simple	No	No	No	No	No	No	No
Q14	Full	No	No	No	No	No	No	No
Q15	Full	No	No	No	No	No	No	No
Q16	Simple	No	No	No	No	No	Yes	No
Q17	Full	No	Yes	No	No	No	Yes	No
Q18	Full	No	Yes	No	No	No	Yes	No
Q19	Full	No	No	No	No	No	No	No

Finally, Figure 30 and Table 9 show us again a set of good execution times gotten by Prolog engines. Its worst results occurred in the queries Q12, Q16, Q17 and Q18. For queries Q17 and Q18 the bad results are probably due to the fact that they are full search (A) with text search (C). Queries Q12 and Q16 are simple search (A) and have the large amount of related data technical feature (G) associated. In the other words, when we analyze why these two queries are different of the others, we can note they are the only queries that gather a large quantity of text. Query Q12 asks for element *body*. This element can have many text data involved. For query Q16, the scenario is worst. This query asks for element *article*. This element contains the element *body* and because of obvious reasons, it contains yet more data.

Although we are querying a multiple document dataset, this fact does not affect the Prolog engines execution times. When the XML files are converted to Prolog facts and rules, they do not stay in separated files in our approach. Thereby, we cannot use this information as possible cause of bad results.

Table 10 shows us the better query mechanism to process queries based on the kind of dataset and the characteristics of the queries. Note that, in Table 10, there is no mention of simple search queries for the “Text Centric Single Document” dataset. This fact happens because there is no simple search query for this dataset in XBench.

In Table 10, we analyze three points to choose the better approach: Type of dataset, type of search and other technical features described earlier. Starting by the data centric single document dataset (DC/SD), we investigate the data generated by the experiment to define which approach we have to use. Looking at Figure 28, we can see that Prolog engines win sixteen times while native tools never win. Because of this, for data centric single document datasets, no matter what is the type of search or other technical features, Prolog engines are always the better approach.

The text centric single document dataset (TC/SD) do not have queries of type simple search. However, some technical features influence the results obtained by each approach. In Figure 29, we can see that XML native tools win nine times while Prolog engines win just seven times. Analyzing this information and also the technical features in Table 8 we can conclude that queries that have at least one of the technical features **sorting triggered by schema** or **quantifier operator** or **large amount of related data**, are more efficiently processed by XML native approaches. For all other technical features, the better approach is Prolog engines.

Finally, we analyze the text centric multiple document dataset (TC/MD). Looking at Figure 30, we can see that Prolog engines win fifteen times while XML native tools win four times. When we investigate the queries with bad results for Prolog engines and analyze Table 9, we conclude that queries with technical feature **large amount of related data** must be executed by XML native tools. In all other cases, Prolog engines have better performance.

Table 10: Heuristic map

Dataset	**Query/Schema**		
Kind of Dataset	**Type of Search**	**Other Technical Features Associated to Query**	**Better Approach To Use**
Data Centric Single Document	Simple or Full	Any	Prolog Engines
Text Centric Single Document	Full	B, E or G	XML Native
		Any, except B, E and G	Prolog Engines
Text Centric Multiple Document	Simple or Full	G	XML Native
		Any, except G	Prolog Engines

All of these results are summarized in Table 10, which presents our heuristic map. When sorting by schema (B) is needed, Prolog engines present loss of performance to deal with the additional processing. This fact can be clearly observed in the text centric single document (TC/SD) results. Queries with intensive text handling (G) also tend to have higher execution times for Prolog engines. This handling can be just a search for a substring or an

attempt to build the final query answer. YAP seems to deal better with this feature, but even so, Sedna is a better solution in general for this situation. Another important thing to observe is that for Simple Search, Prolog engines are usually faster than native XML tools.

In order to be fair, we connected the technical features and kind of dataset with the better approaches. For instance, query Q03 in the single document data centric dataset, has the distinct values (H) technical feature. For this query, SWI-Prolog engine got bad results, but, even so, we did not connected this technical feature as the reason of the bad result because there is only a single occurrence and it was specific for SWI-Prolog.

CHAPTER 5 – CONCLUSION

5.1 FINAL REMARKS

In the course of time, more and more applications and services are getting available on the internet. Hence, the need arose to create a pattern to exchange information between different applications and platforms. Among many options, XML was adopted as a standard approach, recommended by the World Wide Web Consortium (W3C). Therefore, a large quantity of information is available as XML documents on the Web.

With so much information stored in XML documents, the need to extract information from these documents increases. There are many approaches to query these documents, but they are concerned only in extracting explicit information contained therein. In addition, current approaches do not always have satisfactory performance when applied to large volumes of data.

In this work, we evaluated the approach presented by Lima (2012). Lima's approach, XMLInference, makes it possible to extract Prolog facts from XML documents. The experimental evaluation conducted in Lima's work (LIMA, 2012) shows that Logic queries often provide better response times for processing queries when compared to native approaches to query XML documents. This fact was the main motivation to our work. Our goal was then to analyze when Prolog is a better option to process XML queries than XML native approaches.

We set up three research questions to be answered along this work:

RQ1: Can we represent the XQuery queries proposed by XBench as standard Prolog queries and still get the same results?

RQ1 Answer: We answered the first question in Appendix I where we presented the corresponding Prolog query for each XBench query. We compared the results gathered by each query in both approaches, XML native and XMLInference, using a *diff* tool. The results were the same except for some blank spaces and carriage returns. Although this does not mean that all XQuery queries can be translated into Prolog statements, the fact that Prolog is a Turing-complete language, we know for sure that we can always translate an XQuery query into a Prolog statement.

RQ2: Do the translated queries in RQ1 present a better or competitive execution time when compared to native XML query processing engines?

RQ2 Answer: The second question is answered in Chapter 5 when we compare the execution times of Prolog engines with those obtained by XML native tools. As we presented in Chapter 5, in the majority of the evaluated queries, Prolog engines performed better than native XML approaches. Even when XML native approaches had better execution times, the difference between them were not big.

RQ3: When is it a good choice to use Prolog-based approaches to process XQuery queries?

RQ3 Answer: The answer for the third question is the most important of all because it is the main contribution of our work – the heuristic decision map that we presented in Chapter 5. This heuristic map is the answer for RQ3. Looking at the map, the user or system must be able to choose the better option to execute each query.

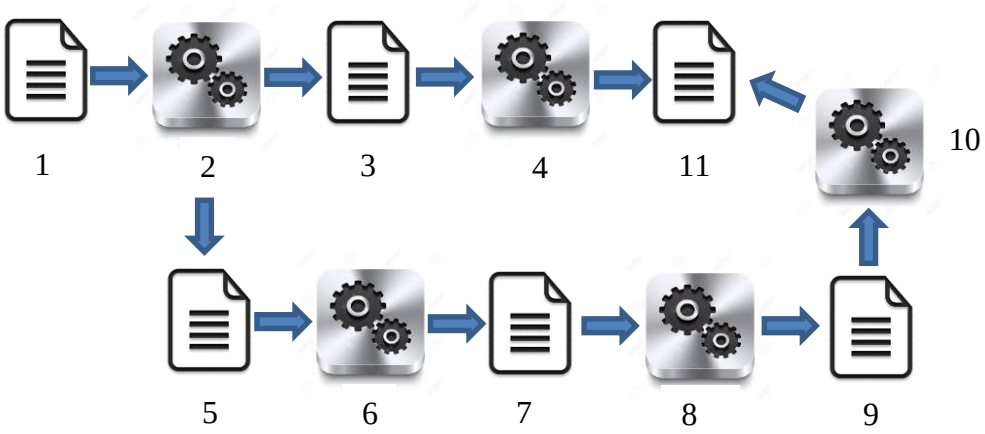
It is important to note that our experiments have several threats to validity. We did not evaluate the query conversion time from XQuery to Prolog. If the conversion time is higher than the execution time, our heuristic would be impacted. Beyond that, the execution times are almost always very low. This fact can penalize approaches whose bootstrap takes longer. In that case, the results could be different for long running queries. We also did not use datasets with different sizes. Thus, our results cannot be generalized to larger or smaller datasets. We also only evaluated a subset of the XML native tools available. There can be a native tool whose results are better than Prolog mechanisms. Finally, the execution times were gathered in different ways for each approach. This may have impacted the results.

During the preparation of this work, we obtained the publication of a paper in a conference (SANTOS; PINHEIRO; BRAGANHOLO, 2012), whose presentation happened in the Brazilian Symposium of Databases in 2012. We are also preparing a journal paper with the results of our work. We will submit this paper as soon as possible.

5.2 FUTURE WORK

Although we have performed several experiments with the aid of XBench, due to time restrictions, it was not possible to evaluate 100% of the benchmark. The translation of

XQuery queries into Prolog rules is a very time consuming step. For now, this task is manual and hard. We evaluated the data centric single document (DC / SD), text centric single document (TC / SD) and text centric multiple document (TC / MD) datasets. The data centric multiple document (DC /MD) dataset was not evaluated, though. However, we believe the results in this dissertation are promising enough, and the DC/MD dataset would result in similar findings. We intend to complete the assessment of the Benchmark and formalize the queries translation in future work.



1- XPath/XQuery
2- Heuristic processor. If XML is a better option go to step 3, otherwise go to step 5
3- Forward XQuery/XPath to native XMLengine
4- Process XQuery/XPath
5- Forward XQuery/XPath to Prolog translator
6- Translate XQuery/XPath to Prolog statement
7- Prolog statement
8- Process query in Prolog engine
9- Prolog result
10- Convert Prolog result to XML
11- XML result

Figure 32: Desired workflow

Another future goal is to complete the workflow as discussed in the related work chapter. In the Figure 32, we can see with details this idea. We plan to develop an approach that is able to receive an XQuery query, apply our heuristic to decide if it's worth converting

it into Prolog. If so, convert it into Prolog and execute it, and finally return the XML query result to the user. Currently, we can use the XMLInference prototype as a preprocessing module that transforms the XML dataset in facts and Prolog rules. The heuristic map constructed in this work can be used to guide the execution flow for the best possible user response time and also works as a motivator for the further development of a complete solution to efficiently process XQuery using Prolog. A key component of this solution, that is still missing, is a module that would be able to convert XPath and XQuery into Prolog, and a module that would be able to transform the results generated by Prolog inference engines into an XML document. This way, XML query processing using Prolog would be completely transparent to the user.

REFERENCES

ALMENDROS-JIMÉNEZ, J.; BECERRA-TERÓN, A.; ENCISO-BANOS, F. J.. Integrating XQuery and Logic Programming. *Applications of Declarative Programming and Knowledge Management*. Heidelberg: Springer-Verlag Berlin, 2007. p. 117–135.

ALMENDROS-JIMÉNEZ, J.; BECERRA-TERÓN, A.; ENCISO-BANOS, F. J.. Querying XML Documents in Logic Programming. *Journal of Theory and Practice of Logic Programming*, v. 8, n. 03, p. 323–361, 2008.

BOAG, S.; CHAMBERLIN, DON; FERNANDEZ, MARY F.; FLORESCU, DANIELA; ROBIE, JONATHAN; SIMÉON, JÉRÔME. *XQuery 1.0: An XML Query Language. W3C Recommendation.* Disponível em: <www.w3.org/TR/xquery>. Acesso em: 9 jun. 2011.

BÖHME, T.; RAHM, E.. XMach-1: A Benchmark for XML Data Management. *Datenbanksysteme in Büro, Technik und Wissenschaft*. [S.l: s.n.], 2001. p. 264–273.

CLARK, J.; DEROSE, S.. *XML Path Language (XPath) Version 1.0*. Disponível em: <www.w3.org/tr/xpath>. Acesso em: 4 abr. 2012.

COSTA, V. S.; ROCHA, R.; DAMAS, L.. The YAP Prolog system. *Theory and Practice of Logic Programming*, v. 12, p. 5–34, jan. 2012.

FOMICHEV, A.; GRINEV, M.; KUZNETSOV, S.. **Sedna: A native XML DBMS**. **Proceedings**... In: International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM). Merin, Czech Republic: 2006.

FREIRE, J; KOOP, D.; SANTOS, E.; SILVA, C.T.. Provenance for Computational Tasks: A Survey. *Computing in Science Engineering*, v. 10, n. 3, p. 11–21, 2008.

JAN WIELEMAKER; TORBJORN LAGER; TOM SCHRIJVERS; MARKUS TRISKA. SWI-Prolog. *Theory and Practice of Logic Programming*, v. 12, n. 1-2, p. 67–96, 2012.

KOTSAKIS, E.. **Structured Information Retrieval in XML Documents**. 2002, Madrid, Spain. *Anais*... Madrid, Spain: [s.n.], 2002. p. 663–667.

LIMA, D.; DELGADO, C.; MURTA, L.; BRAGANHOLO, V.. Towards Querying Implicit Knowledge in XML Documents. *Journal of Information and Data Management (JIDM)*, v. 3, n. 1, p. 51–60, 2012a.

LIMA, D.. *XMLInference: Agregando Inferência à Consultas a Dados XML*. 2012. Dissertação de Mestrado em Informática – Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil, 2012.

LIMA, D.; DELGADO, C.; MURTA, L.; BRAGANHOLO, V.. XMLInference: Consulta a XML Utilizando Inferência. *XMLInference: Consulta a XML utilizando inferência*, 2012b.

MATTOSO, M.; WERNER, CLAUDA; TRAVASSOS, GUILHERME HORTA; BRAGANHOLO, V., OGASA, EDUARDO; OLIVEIRA, DANIEL; CRUZ, SERGIO, MARTINHO, WALLACE; MURTA, L.. Towards Supporting the Life Cycle of Large Scale

Scientific Experiments. *International Journal of Business Process Integration and Management*, v. 5, n. 1, p. 79–92, 2010. Acesso em: 28 ago. 2013.

MEIER, W.. eXist: An Open Source Native XML Database. *Web, Web-Services, and Database Systems*, v. LNCS 2593, p. 169—183, 2003.

RODRIGUES, C.; BRAGANHOLO, V.; MATTOSO, M.. Virtual Partitioning ad-hoc Queries over Distributed XML Databases. *Journal of Information and Data Management (JIDM)*, v. 2, n. 3, p. 495–510, 2011.

RUNAPONGSA, K.; M. PATEL, JIGNESH; JAGADISH, H.V.; CHEN, YUN; AL_KHALIFA, SHURUG. Information Systems. *The Michigan benchmark: Towards XML Query Performance Diagnostics*, p. 73–97, 2006.

SANTOS, F.; PINHEIRO, R.; BRAGANHOLO, V.. **Processamento de Consultas XML usando Máquinas de Inferência**. **Proceedings**... In: Simpósio Brasileiro de Banco de dados. São Paulo, Brasil: 2012.

SCHMIDT, A.*;* WASS, FLORIAN; KERSTEN; MARTIN; FLORESCU, DANIELA; J. CAREY, MICHAEL; MANOLESCU, IONA; BUSSE, RALPH. Why and How to Benchmark XML Databases. *ACM SIGMOD Record*, v. 30, n. 3, p. 27–32, 2001.

SCHMIDT, A.*;* WASS, FLORIAN; KERSTEN; MARTIN; J. CAREY, MICHAEL; MANOLESCU, IONA; BUSSE, RALPH. **XMark: A Benchmark for XML Data Management**. In: International Conference on Very Large Data Bases (VLDB), 2002, Hong Kong, China. *Anais*... Hong Kong, China: [s.n.], 2002. p. 974–985.

SIMÉON, JÉRÔME.*;* FERNÁNDEZ, MAR; CHOI, BYRON; RADHAKRISHNAN, M.; RESNICK, LORI; SUR, GARGI; WADLER, PHILIP. *Galax: An Implementation of XQuery*. [S.l: s.n.], 2000. Disponível em: <http://db.bell-labs.com/galax/>.

YAO, B. B.; OZSU, M. T.; KHANDELWAL, N.. **XBench Benchmark and Performance Testing of XML DBMSs**. In: IEEE International Conference on data Engineering (ICDE). Boston, United States: 2004.

APPENDIX I – XML SCHEMAS OF DATASETS

In this appendix section, we show the XML Schemas for all datastes from XBench (YAO; OZSU; KHANDELWAL, 2004) used in our tests: data centric single document (DC/SD), text centric single document (TC/SD) and text centric multiple document (TC/MD).

1.1 TC/SD SCHEMA

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xs:element name="a" type="xs:string"/>
    <xs:element name="bib">
        <xs:simpleType>
            <xs:restriction base="xs:string"/>
        </xs:simpleType>
    </xs:element>
    <xs:element name="cr" type="xs:IDREF"/>
    <xs:element name="def">
        <xs:complexType mixed="true">
            <xs:choice minOccurs="0" maxOccurs="4">
                <xs:element ref="cr"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:element name="dictionary">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="e" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="e">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="hwg"/>
                <xs:element ref="vfl" minOccurs="0"/>
                <xs:element ref="et" minOccurs="0"/>
                <xs:element ref="ss"/>
            </xs:sequence>
```

```
        <xs:attribute name="id" type="xs:ID"/>
    </xs:complexType>
</xs:element>
<xs:element name="et">
    <xs:complexType>
        <xs:sequence maxOccurs="16">
            <xs:element ref="cr"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="hw">
    <xs:simpleType>
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
</xs:element>
<xs:element name="hwg">
    <xs:complexType>
        <xs:choice maxOccurs="15">
            <xs:element ref="hw"/>
            <xs:element ref="pr" minOccurs="0"/>
            <xs:element ref="pos" minOccurs="0"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
<xs:element name="pos">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="adj."/>
            <xs:enumeration value="adv."/>
            <xs:enumeration value="conj."/>
            <xs:enumeration value="n."/>
            <xs:enumeration value="prep."/>
            <xs:enumeration value="v."/>
            <xs:enumeration value="int."/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
<xs:element name="pr" type="xs:string"/>
<xs:element name="q">
    <xs:complexType>
        <xs:sequence>
```

```
            <xs:element ref="qd"/>
            <xs:element ref="a" minOccurs="0"/>
            <xs:element ref="w"/>
            <xs:element ref="bib" minOccurs="0"/>
            <xs:element ref="loc"/>
            <xs:element ref="qt"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="qd">
    <xs:simpleType>
        <xs:restriction base="xs:short"/>
    </xs:simpleType>
</xs:element>
<xs:element name="qt">
    <xs:complexType mixed="true">
        <xs:choice minOccurs="0" maxOccurs="6">
            <xs:element ref="cr"/>
            <xs:element name="i" type="xs:string"/>
            <xs:element name="b" type="xs:string"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
<xs:element name="loc" type="xs:string"/>
<xs:element name="qp">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="q" maxOccurs="20"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="s">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="def"/>
            <xs:element ref="qp"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ss">
    <xs:complexType>
```

```
            <xs:sequence>
                <xs:element ref="s" maxOccurs="10"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="vd">
        <xs:simpleType>
            <xs:restriction base="xs:string"/>
        </xs:simpleType>
    </xs:element>
    <xs:element name="vf">
        <xs:simpleType>
            <xs:restriction base="xs:string"/>
        </xs:simpleType>
    </xs:element>
    <xs:element name="vfl">
        <xs:complexType>
            <xs:choice maxOccurs="45">
                <xs:element ref="vd" minOccurs="0"/>
                <xs:element ref="vf" maxOccurs="4"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:element name="w" type="xs:string"/>
</xs:schema>
```

1.2 TC/MD SCHEMA

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
<xs:element name="a_id" type="xs:string"/>
<xs:element name="abstract">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="p" maxOccurs="7"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="article">
<xs:complexType>
```

```
    <xs:sequence>
    <xs:element ref="prolog"/>
    <xs:element ref="body"/>
    <xs:element name="epilog">
    <xs:complexType>
        <xs:sequence>
        <xs:element name="acknowledgements" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                <xs:element ref="pa" maxOccurs="3"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="references" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                <xs:element ref="a_id" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        </xs:sequence>
    </xs:complexType>
    </xs:element>
    </xs:sequence>
    <xs:attribute name="id" type="xs:byte" use="required"/>
    <xs:attribute name="lang" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="author">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="name"/>
            <xs:element ref="contact"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="authors">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="author" maxOccurs="48"/>
        </xs:sequence>
```

```
    </xs:complexType>
</xs:element>
<xs:element name="body">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="abstract" minOccurs="0"/>
            <xs:element ref="section" maxOccurs="15"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="section">
    <xs:complexType>
    <xs:sequence>
        <xs:element ref="p" minOccurs="0" maxOccurs="29"/>
        <xs:element ref="subsec" minOccurs="0" maxOccurs="23"/>
    </xs:sequence>
    <xs:attribute name="heading" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="city" type="xs:string"/>
<xs:element name="contact">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="email" minOccurs="0"/>
            <xs:element ref="phone" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="country" type="xs:string"/>
<xs:element name="date" type="xs:date"/>
<xs:element name="dateline">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="city"/>
            <xs:element ref="country"/>
            <xs:element ref="date"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="email">
    <xs:simpleType>
```

```
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
</xs:element>
<xs:element name="genre" type="xs:string"/>
<xs:element name="keyword" type="xs:string"/>
<xs:element name="keywords">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="keyword" maxOccurs="19"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="name" type="xs:string"/>
<xs:element name="p" type="xs:string"/>
<xs:element name="pa" type="xs:string"/>
<xs:element name="phone" type="xs:string"/>
<xs:element name="prolog">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="title"/>
            <xs:element ref="authors" minOccurs="0"/>
            <xs:element ref="dateline" minOccurs="0"/>
            <xs:element ref="genre" minOccurs="0"/>
            <xs:element ref="keywords" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="subsec">
<xs:complexType>
<xs:sequence>
    <xs:element ref="p" minOccurs="0" maxOccurs="46"/>
    <xs:element name="subsec" minOccurs="0" maxOccurs="31">
    <xs:complexType>
    <xs:sequence>
    <xs:element ref="p" minOccurs="0" maxOccurs="11"/>
    <xs:element name="subsec" minOccurs="0" maxOccurs="8">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="p" maxOccurs="12"/>
            </xs:sequence>
            <xs:attribute name="heading" type="xs:string"
```

```
            use="required"/>
        </xs:complexType>
    </xs:element>
    </xs:sequence>
    <xs:attribute name="heading" type="xs:string" use="required"/>
    </xs:complexType>
    </xs:element>
</xs:sequence>
<xs:attribute name="heading" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="title" type="xs:string"/>
</xs:schema>
```

1.3 DC/SD SCHEMA

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xs:element name="FAX_number" type="xs:string"/>
    <xs:element name="ISBN" type="xs:string"/>
    <xs:element name="attributes">
        <xs:complexType>
            <xs:all>
                <xs:element ref="ISBN"/>
                <xs:element ref="number_of_pages"/>
                <xs:element ref="type_of_book"/>
                <xs:element ref="size_of_book"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
    <xs:element name="author">
        <xs:complexType>
            <xs:all>
                <xs:element name="name">
                    <xs:complexType>
                        <xs:all>
                            <xs:element ref="first_name"/>
                            <xs:element ref="middle_name"/>
                            <xs:element ref="last_name"/>
```

```
                    </xs:all>
                </xs:complexType>
            </xs:element>
            <xs:element ref="date_of_birth"/>
            <xs:element ref="biography"/>
            <xs:element name="contact_information">
                <xs:complexType>
                    <xs:all>
                        <xs:element name="mailing_address">
                            <xs:complexType>
                            <xs:all>
                            <xs:element ref="street_information"/>
                            <xs:element ref="name_of_city"/>
                            <xs:element ref="name_of_state"/>
                            <xs:element ref="zip_code"/>
                            <xs:element name="name_of_country"
                                type="xs:string"/>
                            </xs:all>
                            </xs:complexType>
                        </xs:element>
                        <xs:element ref="phone_number"/>
                        <xs:element ref="email_address"/>
                    </xs:all>
                </xs:complexType>
            </xs:element>
        </xs:all>
    </xs:complexType>
</xs:element>
<xs:element name="authors">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="author" maxOccurs="4"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="biography" type="xs:string"/>
<xs:element name="catalog">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="item" maxOccurs="unbounded"/>
        </xs:sequence>
```

```
    </xs:complexType>
</xs:element>
<xs:element name="cost">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:decimal">
                <xs:attribute name="currency" type="xs:string"
                    use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="country">
    <xs:complexType>
        <xs:all>
            <xs:element name="name" type="xs:string"/>
            <xs:element ref="exchange_rate"/>
            <xs:element ref="currency"/>
        </xs:all>
    </xs:complexType>
</xs:element>
<xs:element name="currency" type="xs:string"/>
<xs:element name="data" type="xs:string"/>
<xs:element name="date_of_birth" type="xs:date"/>
<xs:element name="date_of_release" type="xs:date"/>
<xs:element name="description" type="xs:string"/>
<xs:element name="email_address" type="xs:string"/>
<xs:element name="web_site" type="xs:string"/>
<xs:element name="exchange_rate" type="xs:decimal"/>
<xs:element name="first_name" type="xs:string"/>
<xs:element name="height">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:decimal">
                <xs:attribute name="unit" type="xs:string"
                    use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="image">
```

```
    <xs:complexType>
        <xs:all>
            <xs:element ref="data"/>
        </xs:all>
    </xs:complexType>
</xs:element>
<xs:element name="item">
    <xs:complexType>
        <xs:all>
            <xs:element ref="title"/>
            <xs:element ref="authors"/>
            <xs:element ref="date_of_release"/>
            <xs:element ref="publisher"/>
            <xs:element ref="subject"/>
            <xs:element ref="description"/>
            <xs:element ref="related_items"/>
            <xs:element ref="media"/>
            <xs:element ref="pricing"/>
            <xs:element ref="attributes"/>
        </xs:all>
        <xs:attribute name="id" type="xs:ID" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="item_id" type="xs:IDREF"/>
<xs:element name="last_name" type="xs:string"/>
<xs:element name="length">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:decimal">
                <xs:attribute name="unit" type="xs:string"
                    use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="media">
    <xs:complexType>
        <xs:all>
            <xs:element ref="thumbnail"/>
            <xs:element ref="image"/>
        </xs:all>
```

```
    </xs:complexType>
</xs:element>
<xs:element name="middle_name" type="xs:string"/>
<xs:element name="name_of_city" type="xs:string"/>
<xs:element name="name_of_country" type="xs:string"/>
<xs:element name="name_of_state" type="xs:string"/>
<xs:element name="number_of_pages" type="xs:short"/>
<xs:element name="phone_number" type="xs:string"/>
<xs:element name="pricing">
    <xs:complexType>
        <xs:all>
            <xs:element ref="suggested_retail_price"/>
            <xs:element ref="cost"/>
            <xs:element ref="when_is_available"/>
            <xs:element ref="quantity_in_stock"/>
        </xs:all>
    </xs:complexType>
</xs:element>
<xs:element name="publisher">
    <xs:complexType>
        <xs:all>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="contact_information">
                <xs:complexType>
                    <xs:all>
                        <xs:element name="mailing_address">
                            <xs:complexType>
                            <xs:all>
                            <xs:element ref="street_information"/>
                            <xs:element ref="name_of_city"/>
                            <xs:element ref="name_of_state"/>
                            <xs:element ref="zip_code"/>
                            <xs:element ref="country"/>
                            </xs:all>
                            </xs:complexType>
                        </xs:element>
                        <xs:element ref="FAX_number"
                            minOccurs="0"/>
                        <xs:element ref="phone_number"/>
                        <xs:element ref="web_site"/>
                    </xs:all>
```

```
                </xs:complexType>
            </xs:element>
        </xs:all>
    </xs:complexType>
</xs:element>
<xs:element name="quantity_in_stock" type="xs:byte"/>
<xs:element name="related_item">
    <xs:complexType>
        <xs:all>
            <xs:element ref="item_id"/>
        </xs:all>
    </xs:complexType>
</xs:element>
<xs:element name="related_items">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="related_item" minOccurs="0"
                maxOccurs="5"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="size_of_book">
    <xs:complexType>
        <xs:all>
            <xs:element ref="length"/>
            <xs:element ref="width"/>
            <xs:element ref="height"/>
        </xs:all>
    </xs:complexType>
</xs:element>
<xs:element name="street_address" type="xs:string"/>
<xs:element name="street_information">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="street_address" maxOccurs="2"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="subject" type="xs:string"/>
<xs:element name="suggested_retail_price">
    <xs:complexType>
```

```
            <xs:simpleContent>
                <xs:extension base="xs:decimal">
                    <xs:attribute name="currency" type="xs:string"
                        use="required"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="thumbnail">
        <xs:complexType>
            <xs:all>
                <xs:element ref="data"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
    <xs:element name="title" type="xs:string"/>
    <xs:element name="type_of_book" type="xs:string"/>
    <xs:element name="when_is_available" type="xs:date"/>
    <xs:element name="width">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:decimal">
                    <xs:attribute name="unit" type="xs:string"
                        use="required"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="zip_code" type="xs:string"/>
</xs:schema>
```

APPENDIX II – BENCHMARK QUERIES

As mentioned before, we used the XQuery queries provided by XBench in our tests. In this chapter, we briefly describe these queries and show the corresponding prolog queries developed by us.

1.1 SINGLE DOCUMENT DATA CENTRIC XQUERY QUERIES

In this section, we present the XQuery data centric queries and their matching Prolog queries.

- **Q1** - Return the item that has matching item id attribute value (I1).
  - **XQuery**

```
for $item in input()/catalog/:item[@id="I1"]
return
   $item
```

  - **Prolog**

```
pConsultaUm :- pItem('I1').

pAttributes(ID) :- attributes(ID, ATT_ID), isbn(ATT_ID,_,ISBN),
print('<attributes>'), print('< ISBN>'), print(ISBN), print('</
ISBN>'), nl, number_of_pages(ATT_ID,_,NPAGES),
print('<number_of_pages>'), print(NPAGES),
print('</number_of_pages>'), nl, type_of_book(ATT_ID,_,TBOOK),
print('<type_of_book>'), print(TBOOK), print('</type_of_book>'), nl,
size_of_book(ATT_ID,SBOOK_ID), length(SBOOK_ID, UNIT_LENGTH_ID,
LENGTH), unit(UNIT_LENGTH_ID,_,UNIT_LENGTH), print('<size_of_book>'),
print('<length unit="'), print(UNIT_LENGTH), print('">'),
print(LENGTH), print('</length>'), nl, width(SBOOK_ID, UNIT_WIDTH_ID,
WIDTH), unit(UNIT_WIDTH_ID,_,UNIT_WIDTH), print('<width unit="'),
print(UNIT_WIDTH), print('">'), print(WIDTH), print('</width>'), nl,
height(SBOOK_ID, UNIT_HEIGHT_ID,HEIGHT),
unit(UNIT_HEIGHT_ID,_,UNIT_HEIGHT), print('<height unit="'),
print(UNIT_HEIGHT), print('">'), print(HEIGHT), print('</height>'),
nl, print('</size_of_book>'), nl, print('</attributes>'), nl.
```

```
pMedia(ID) :- media(ID,MEDIA_ID), thumbnail(MEDIA_ID, THUMB_ID),
data(THUMB_ID, _,DATA_THUMB), print('<media> <thumbnail><data>'),
print(DATA_THUMB), print('</data></thumbnail>'), nl, image(MEDIA_ID,
IMG_ID), data(IMG_ID,_,DATA_IMG), print(' <image><data>'),
print(DATA_IMG), print('</data></image></media>'), nl.

pPricing(ID) :- pricing(ID, PRICE_ID),
suggested_retail_price(PRICE_ID, CURR_ID, PRICE),
currency(CURR_ID,_,CURRENCY), print('<pricing>'), print('<
suggested_retail_price currency="'), print(CURRENCY), print('">'),
print(PRICE), print('</ suggested_retail_price>'), nl, cost(PRICE_ID,
CURR_COST_ID, COST), currency(CURR_COST_ID,_,CURRENCY_COST),
print('<cost currency="'), print(CURRENCY_COST), print('">'),
print(COST), print('</cost>'), nl,
when_is_available(PRICE_ID,_,AVAILABLE), print('<
when_is_available>'), print(AVAILABLE), print('</
when_is_available>'), nl, quantity_in_stock(PRICE_ID,_,QT),
print('<quantity _in_stock>'), print(QT), print('</quantity
_in_stock>'), print('</pricing>') nl.

pMailingAddressPublisher(ID) :- mailing_address(ID,MAIL_INFO),
print('<mailing_address>'), pStreetInfo(MAIL_INFO),
name_of_city(MAIL_INFO,_,NCITY), print('<name_of_city>'),
print(NCITY), print('</name_of_city>'), nl,
name_of_state(MAIL_INFO,_,NSTATE), print('<name_of_state>'),
print(NSTATE), print('</name_of_state>'), nl,
zip_code(MAIL_INFO,_,ZCODE), print('<zip_code>'), print(ZCODE),
print('</zip_code>'), nl, country(MAIL_INFO,COUNTRY_ID),
name(COUNTRY_ID,_,NCOUNTRY), print('<country><name>'),
print(NCOUNTRY), print('</name>'), nl,
exchange_rate(COUNTRY_ID,_,ERATE), print('<exchange_rate>'),
print(ERATE), print('<exchange_rate>'), nl,
currency(COUNTRY_ID,_,CURR), print('<currency>'), print(CURR),
print('</currency></country>'), print('</mailing_address>'), nl.

pContactInfoPublisher(ID) :- contact_information(ID,CINFO),
print('<contact_information>'), pMailingAddressPublisher(CINFO),
phone_number(CINFO,_,NUMBER), print('<phone_number>'), print(NUMBER),
print('</phone_number>'), nl, web_site(CINFO,_,URL), print('<web_site
>'), print(URL), print('</web_site >'), nl,
print('</contact_information>'), nl.
```

```
pStreetInfo_1([]).
pStreetInfo_1([H|T]) :- print('<street_address>'), print(H),
print('</street_address>'), nl, pStreetInfo_1(T).
pStreetInfo(ID) :- street_information(ID,X),
print('<street_information>'),
findall(Y,street_address(X,_,Y),LISTA), pStreetInfo_1(LISTA),
print('</street_information>'),.

pMailingAddress(ID) :- mailing_address(ID,MAIL_INFO),
print('<mailing_address>'), pStreetInfo(MAIL_INFO),
name_of_city(MAIL_INFO,_,NCITY), print('<name_of_city>'),
print(NCITY), print('</name_of_city>'), nl,
name_of_state(MAIL_INFO,_,NSTATE), print('<name_of_state>'),
print(NSTATE), print('</name_of_state>'), nl,
zip_code(MAIL_INFO,_,ZCODE), print('<zip_code>'), print(ZCODE),
print('</zip_code>'), nl, name_of_country(MAIL_INFO,_,NCOUNTRY),
print('<name_of_country>'), print(NCOUNTRY),
print('</name_of_country>'), nl, print('</mailing_address>'), nl.

pContactInfo(ID) :- contact_information(ID,CINFO),
print('<contact_information>'), pMailingAddress(CINFO),
phone_number(CINFO,_,NUMBER), print('<phone_number>'), print(NUMBER),
print('</phone_number>'), nl, email_address(CINFO,_,EMAIL),
print('<email_address>'), print(EMAIL), print('</email_address>'),
nl, print('</contact_information>'), nl.

pPublisher(ID) :- publisher(ID,PUB), name(PUB,_,NAME),
rint('<publisher>'), print('<name>'), print(NAME), print('</name>'),
nl, pContactInfoPublisher(PUB), print('</publisher>').

pName(ID) :-  name(ID,NAME_ID), print('<name>'),
first_name(NAME_ID,_,NAME), print('<first_name>'), print(NAME),
print('</first_name>'), nl, middle_name(NAME_ID,_,MNAME),
print('<middle_name>'), print(MNAME), print('</middle_name>'), nl,
last_name(NAME_ID,_,LNAME), print('<last_name>'), print(LNAME),
print('</last_name>'), nl, print('</name>'), nl.

pAuthors([]).
pAuthors([H|T]) :- print('<author>'), pName(H),
date_of_birth(H,_,DATE), print('<date_of_birth>'), print(DATE),
```

```
print('</date_of_birth>'), nl, biography(H,_,BIO),
print('<biography>'), print(BIO), print('</biography>'), nl,
pContactInfo(H), print('</author>'), nl, pAuthors(T).

pRelItens([]).
pRelItens([H|T]) :- item_id(H,_,ITEM), print('<related_item>'),
print('<item_id>'), print(ITEM), print('</item_id>'), nl,
print('</related_item>'), pRelItens(T).

dumpRelItens(ID) :- related_items(ID,REL_ITENS_ID),
findall(X,related_item(REL_ITENS_ID,X),REL_ITENS_LIST),
print('<related_items>'), nl, pRelItens(REL_ITENS_LIST), nl,
print('</related_items>').

dumpAuthors(ID) :- authors(ID,AUTHORS_ID),
findall(X,author(AUTHORS_ID,X),AUTHORS_LIST), print('<authors>'), nl,
pAuthors(AUTHORS_LIST), print('</authors>').

pItem(IID) :- id(ID,_,IID), print('<item id="'), print(IID),
print('">'), title(ID,_,TITLE), print('<title>'), print(TITLE),
print('</title>'), nl, dumpAuthors(ID),
date_of_release(ID,_,DT_RELEASE), print('<date_of_release>'),
print(DT_RELEASE), print('</date_of_release>'), nl, pPublisher(ID),
subject(ID,_,SUBJECT), print('<subject>'), print(SUBJECT),
print('/subject>'), nl, description(ID,_,DESC),
print('<description>'), print(DESC), print('</description>'), nl,
dumpRelItens(ID), pMedia(ID), pPricing(ID), pAttributes(ID),
print('</item>').
```

- **Q2** - Find the title of the item, which has matching author first name (Ben).
  - **XQuery**

```
for $item in input()/catalog/:item
where $item/authors/author/name/first_name = "Ben"
return
   $item/title
```

  - **Prolog**

```
pConsultaDois :- findall(X,first_name(X,_,'Ben'),LISTA),
```

```
pConsultaDois_1(LISTA).
pConsultaDois_1([]).
pConsultaDois_1([H|T]) :- name(Y,H), author(Z,Y), authors(A,Z),
title(A,_,TITLE), print('<title>'), print(TITLE), print('</title>'),
nl, pConsultaDois_1(T).
```

- **Q3** - Group items released in a certain year (1990), by publisher name and calculate the total number of items for each group.

  - **XQuery**

```
for $a in distinct-values (input()/catalog/:item
[date_of_release >= "1990-01-01"]
[date_of_release < "1991-01-01"]/publisher/name)
let $b := input()/catalog/:item/publisher[name=$a]
return
   <Output>
      <Publisher>{$a/text()}</Publisher>
      <NumberOfItems>{count($b)}</NumberOfItems>
   </Output>
```

  - **Prolog**

```
pConsultaTres_1([]).

pConsultaTres_1([H|T]) :-
findall(Y,(name(X,Y,H),publisher(_,X)),LISTA), length(LISTA,SIZE),
print('<Output>'), nl, print('<Publisher>'), print(H),
print('</Publisher>'), nl, print('<NumberOfItens>'), print(SIZE),
print('</NumberOfItens>'), nl, print('</Output>'), nl,
pConsultaTres_1(T).

pConsultaTres :- setof(W,X^Y^Z^(publisher(X,Y),name(Y,Z,W)),NAMES),
maiores_c4(NAMES,'1990-01-01',NAMES_2), menores_c4(NAMES_2,'1991-01-
01',NAMES_3), pConsultaTres_1(NAMES_3).

maiores_c4([],_,[]).
maiores_c4([E|L],D,K) :- name(_1,_,E),publisher(_2,_1),
date_of_release(_2,_,DT_E), (DT_E @< D), maiores_c4(L,D,K).
maiores_c4([E|L],D,[E|K]) :- name(_1,_,E),publisher(_2,_1),
date_of_release(_2,_,DT_E), (DT_E @>= D) , maiores_c4(L,D,K).
```

```
menores_c4([],_,[]).
menores_c4([E|L],D,K) :- name(_1,_,E),publisher(_2,_1),
date_of_release(_2,_,DT_E), (DT_E @> D), menores_c4(L,D,K).
menores_c4([E|L],D,[E|K]) :- name(_1,_,E),publisher(_2,_1),
date_of_release(_2,_,DT_E), (DT_E @=< D) , menores_c4(L,D,K).
```

- **Q4** - List the item id of the previous item of a matching item with id attribute value (I2).

  - **XQuery**

```
let $item := input()/catalog/:item[@id="I2"]
for  $prevItem in input()/catalog/:item
[. << $item][position() = last()]
return
   <Output>
      <CurrentItem>{$item/@id}</CurrentItem>
      <PreviousItem>{$prevItem/@id}</PreviousItem>
   </Output>
```

  - **Prolog**

```
pConsultaQuatro :- findall(Y,id(X,_,Y),LISTA), print('<Output>'), nl,
pConsultaQuatro_1(LISTA,'No Previous'), print('</Output>').
pConsultaQuatro_1([],_).
pConsultaQuatro_1([H|T],K) :- H = 'I2', print('<PreviousItem>'),
print(H), print('</PreviousItem>'), nl, print('<CurrentItem>'),
print(K), print('</CurrentItem>').
pConsultaQuatro_1([H|T],K) :- not(H = 'I2'), pConsultaQuatro_1(T,H).
```

- **Q5** - Return the information about the first author of item with a matching id attribute value (I3).

  - **XQuery**

```
for $a in input()/catalog/:item[@id="I3"]
return
   $a/authors/author[1]
```

  - **Prolog**

```
pConsultaCinco :- id(_1,_,'I3'), authors(_1,_2), author(_2,_3),
pAuthors([_3|[]]).
```

- **Q6** - Return item information where some authors are from certain country (Canada).
  - **XQuery**

```
for $item in input()/catalog/:item
where some $auth in
   $item/authors/author/contact_information/mailing_address
satisfies $auth/name_of_country = "Canada"
return
    $item
```

  - **Prolog**

```
pConsultaSeis :- findall(Y,authors(X,Y),LISTA),
pConsultaSeis_1(LISTA).

pConsultaSeis_1([]).
pConsultaSeis_1([H|T]) :- findall(Y,author(H,Y),LISTA),
pConsultaSeis_2(LISTA), pConsultaSeis_1(T).

pConsultaSeis_2([]).
pConsultaSeis_2([H|T]) :- contact_information(H,_1),
mailing_address(_1,_2), name_of_country(_2,_,_3), (_3 = 'Canada'),
author(_4, H), authors(_5, _4), id(_5,_,_6), pItem(_6), !.
pConsultaSeis_2([H|T]) :- pConsultaSeis_2(T).

pAttributes(ID) :- attributes(ID, ATT_ID), isbn(ATT_ID,_,ISBN),
print('<attributes>'), print('< ISBN>'), print(ISBN), print('</
ISBN>'), nl, number_of_pages(ATT_ID,_,NPAGES),
print('<number_of_pages>'), print(NPAGES),
print('</number_of_pages>'), nl, type_of_book(ATT_ID,_,TBOOK),
print('<type_of_book>'), print(TBOOK), print('</type_of_book>'), nl,
size_of_book(ATT_ID,SBOOK_ID), length(SBOOK_ID, UNIT_LENGTH_ID,
LENGTH), unit(UNIT_LENGTH_ID,_,UNIT_LENGTH), print('<size_of_book>'),
print('<length unit="'), print(UNIT_LENGTH), print('">'),
print(LENGTH), print('</length>'), nl, width(SBOOK_ID, UNIT_WIDTH_ID,
WIDTH), unit(UNIT_WIDTH_ID,_,UNIT_WIDTH), print('<width unit="'),
print(UNIT_WIDTH), print('">'), print(WIDTH), print('</width>'), nl,
```

```
height(SBOOK_ID, UNIT_HEIGHT_ID,HEIGHT),
unit(UNIT_HEIGHT_ID,_,UNIT_HEIGHT), print('<height unit="'),
print(UNIT_HEIGHT), print('">'), print(HEIGHT), print('</height>'),
nl, print('</size_of_book>'), nl, print('</attributes>'), nl.

pMedia(ID) :- media(ID,MEDIA_ID), thumbnail(MEDIA_ID, THUMB_ID),
data(THUMB_ID, _,DATA_THUMB), print('<media> <thumbnail><data>'),
print(DATA_THUMB), print('</data></thumbnail>'), nl, image(MEDIA_ID,
IMG_ID), data(IMG_ID,_,DATA_IMG), print(' <image><data>'),
print(DATA_IMG), print('</data></image></media>'), nl.

pPricing(ID) :- pricing(ID, PRICE_ID),
suggested_retail_price(PRICE_ID, CURR_ID, PRICE),
currency(CURR_ID,_,CURRENCY), print('<pricing>'), print('<
suggested_retail_price currency="'), print(CURRENCY), print('">'),
print(PRICE), print('</ suggested_retail_price>'), nl, cost(PRICE_ID,
CURR_COST_ID, COST), currency(CURR_COST_ID,_,CURRENCY_COST),
print('<cost currency="'), print(CURRENCY_COST), print('"'),
print(COST), print('</cost>'), nl,
when_is_available(PRICE_ID,_,AVAILABLE), print('<
when_is_available>'), print(AVAILABLE), print('</
when_is_available>'), nl, quantity_in_stock(PRICE_ID,_,QT),
print('<quantity _in_stock>'), print(QT), print('</quantity
_in_stock>'), print('</pricing>') nl.

pMailingAddressPublisher(ID) :- mailing_address(ID,MAIL_INFO),
print('<mailing_address>'), pStreetInfo(MAIL_INFO),
name_of_city(MAIL_INFO,_,NCITY), print('<name_of_city>'),
print(NCITY), print('</name_of_city>'), nl,
name_of_state(MAIL_INFO,_,NSTATE), print('<name_of_state>'),
print(NSTATE), print('</name_of_state>'), nl,
zip_code(MAIL_INFO,_,ZCODE), print('<zip_code>'), print(ZCODE),
print('</zip_code>'), nl, country(MAIL_INFO,COUNTRY_ID),
name(COUNTRY_ID,_,NCOUNTRY), print('<country><name>'),
print(NCOUNTRY), print('</name>'), nl,
exchange_rate(COUNTRY_ID,_,ERATE), print('<exchange_rate>'),
print(ERATE), print('<exchange_rate>'), nl,
currency(COUNTRY_ID,_,CURR), print('<currency>'), print(CURR),
print('</currency></country>'), print('</mailing_address>'), nl.

pContactInfoPublisher(ID) :- contact_information(ID,CINFO),
```

```
print('<contact_information>'), pMailingAddressPublisher(CINFO),
phone_number(CINFO,_,NUMBER), print('<phone_number>'), print(NUMBER),
print('</phone_number>'), nl, web_site(CINFO,_,URL), print('<web_site
>'), print(URL), print('</web_site >'), nl,
print('</contact_information>'), nl.

pStreetInfo_1([]).
pStreetInfo_1([H|T]) :- print('<street_address>'), print(H),
print('</street_address>'), nl, pStreetInfo_1(T).
pStreetInfo(ID) :- street_information(ID,X),
print('<street_information>'),
findall(Y,street_address(X,_,Y),LISTA), pStreetInfo_1(LISTA),
print('</street_information>'),.

pMailingAddress(ID) :- mailing_address(ID,MAIL_INFO),
print('<mailing_address>'), pStreetInfo(MAIL_INFO),
name_of_city(MAIL_INFO,_,NCITY), print('<name_of_city>'),
print(NCITY), print('</name_of_city>'), nl,
name_of_state(MAIL_INFO,_,NSTATE), print('<name_of_state>'),
print(NSTATE), print('</name_of_state>'), nl,
zip_code(MAIL_INFO,_,ZCODE), print('<zip_code>'), print(ZCODE),
print('</zip_code>'), nl, name_of_country(MAIL_INFO,_,NCOUNTRY),
print('<name_of_country>'), print(NCOUNTRY),
print('</name_of_country>'), nl, print('</mailing_address>'), nl.

pContactInfo(ID) :- contact_information(ID,CINFO),
print('<contact_information>'), pMailingAddress(CINFO),
phone_number(CINFO,_,NUMBER), print('<phone_number>'), print(NUMBER),
print('</phone_number>'), nl, email_address(CINFO,_,EMAIL),
print('<email_address>'), print(EMAIL), print('</email_address>'),
nl, print('</contact_information>'), nl.

pPublisher(ID) :- publisher(ID,PUB), name(PUB,_,NAME),
rint('<publisher>'), print('<name>'), print(NAME), print('</name>'),
nl, pContactInfoPublisher(PUB), print('</publisher>').

pName(ID) :-  name(ID,NAME_ID), print('<name>'),
first_name(NAME_ID,_,NAME), print('<first_name>'), print(NAME),
print('</first_name>'), nl, middle_name(NAME_ID,_,MNAME),
print('<middle_name>'), print(MNAME), print('</middle_name>'), nl,
last_name(NAME_ID,_,LNAME), print('<last_name>'), print(LNAME),
```

```
print('</last_name>'), nl, print('</name>'), nl.

pAuthors([]).
pAuthors([H|T]) :- print('<author>'), pName(H),
date_of_birth(H,_,DATE), print('<date_of_birth>'), print(DATE),
print('</date_of_birth>'), nl, biography(H,_,BIO),
print('<biography>'), print(BIO), print('</biography>'), nl,
pContactInfo(H), print('</author>'), nl, pAuthors(T).

pRelItens([]).
pRelItens([H|T]) :- item_id(H,_,ITEM), print('<related_item>'),
print('<item_id>'), print(ITEM), print('</item_id>'), nl,
print('</related_item>'), pRelItens(T).

dumpRelItens(ID) :- related_items(ID,REL_ITENS_ID),
findall(X,related_item(REL_ITENS_ID,X),REL_ITENS_LIST),
print('<related_items>'), nl, pRelItens(REL_ITENS_LIST), nl,
print('</related_items>').

dumpAuthors(ID) :- authors(ID,AUTHORS_ID),
findall(X,author(AUTHORS_ID,X),AUTHORS_LIST), print('<authors>'), nl,
pAuthors(AUTHORS_LIST), print('</authors>').

pItem(IID) :- id(ID,_,IID), print('<item id="'), print(IID),
print('">'), title(ID,_,TITLE), print('<title>'), print(TITLE),
print('</title>'), nl, dumpAuthors(ID),
date_of_release(ID,_,DT_RELEASE), print('<date_of_release>'),
print(DT_RELEASE), print('</date_of_release>'), nl, pPublisher(ID),
subject(ID,_,SUBJECT), print('<subject>'), print(SUBJECT),
print('/subject>'), nl, description(ID,_,DESC),
print('<description>'), print(DESC), print('</description>'), nl,
dumpRelItens(ID), pMedia(ID), pPricing(ID), pAttributes(ID),
print('</item>').
```

- **Q7** - Return item information where all its authors are from certain country (Canada).
  - **XQuery**

```
for $item in input()/catalog/:item
where every $add in
   $item/authors/author/contact_information/mailing_address
```

```
satisfies $add/name_of_country = "Canada"
return
    $item
```

- **Prolog**

```
everyC7([]).
everyC7([H2|T2]) :- contact_information(H2,_1),
mailing_address(_1,_2), name_of_country(_2,_,_3), _3 = 'Canada',
everyC7(T2).

pConsultaSete :- findall(Y,authors(X,Y),LISTA),
pConsultaSete_1(LISTA).
pConsultaSete_1([]).
pConsultaSete_1([H|T]) :- findall(Y,author(H,Y),LISTA_1),
pConsultaSete_2(LISTA_1), pConsultaSete_1(T).
pConsultaSete_2([]).
pConsultaSete_2([H1|T1]) :- everyC7([H1|T1]), author(_1, H1),
authors(_2, _1), id(_2,_,_3), pItem(_3).
pConsultaSete_2([H1|T1]) :- not(everyC7([H1|T1])).

pAttributes(ID) :- attributes(ID, ATT_ID), isbn(ATT_ID,_,ISBN),
print('<attributes>'), print('< ISBN>'), print(ISBN), print('</
ISBN>'), nl, number_of_pages(ATT_ID,_,NPAGES),
print('<number_of_pages>'), print(NPAGES),
print('</number_of_pages>'), nl, type_of_book(ATT_ID,_,TBOOK),
print('<type_of_book>'), print(TBOOK), print('</type_of_book>'), nl,
size_of_book(ATT_ID,SBOOK_ID), length(SBOOK_ID, UNIT_LENGTH_ID,
LENGTH), unit(UNIT_LENGTH_ID,_,UNIT_LENGTH), print('<size_of_book>'),
print('<length unit="'), print(UNIT_LENGTH), print('">'),
print(LENGTH), print('</length>'), nl, width(SBOOK_ID, UNIT_WIDTH_ID,
WIDTH), unit(UNIT_WIDTH_ID,_,UNIT_WIDTH), print('<width unit="'),
print(UNIT_WIDTH), print('">'), print(WIDTH), print('</width>'), nl,
height(SBOOK_ID, UNIT_HEIGHT_ID,HEIGHT),
unit(UNIT_HEIGHT_ID,_,UNIT_HEIGHT), print('<height unit="'),
print(UNIT_HEIGHT), print('">'), print(HEIGHT), print('</height>'),
nl, print('</size_of_book>'), nl, print('</attributes>'), nl.

pMedia(ID) :- media(ID,MEDIA_ID), thumbnail(MEDIA_ID, THUMB_ID),
data(THUMB_ID, _,DATA_THUMB), print('<media> <thumbnail><data>'),
print(DATA_THUMB), print('</data></thumbnail>'), nl, image(MEDIA_ID,
```

```
IMG_ID), data(IMG_ID,_,DATA_IMG), print(' <image><data>'),
print(DATA_IMG), print('</data></image></media>'), nl.

pPricing(ID) :- pricing(ID, PRICE_ID),
suggested_retail_price(PRICE_ID, CURR_ID, PRICE),
currency(CURR_ID,_,CURRENCY), print('<pricing>'), print('<
suggested_retail_price currency="'), print(CURRENCY), print('">'),
print(PRICE), print('</ suggested_retail_price>'), nl, cost(PRICE_ID,
CURR_COST_ID, COST), currency(CURR_COST_ID,_,CURRENCY_COST),
print('<cost currency="'), print(CURRENCY_COST), print('"'),
print(COST), print('</cost>'), nl,
when_is_available(PRICE_ID,_,AVAILABLE), print('<
when_is_available>'), print(AVAILABLE), print('</
when_is_available>'), nl, quantity_in_stock(PRICE_ID,_,QT),
print('<quantity _in_stock>'), print(QT), print('</quantity
_in_stock>'), print('</pricing>') nl.

pMailingAddressPublisher(ID) :- mailing_address(ID,MAIL_INFO),
print('<mailing_address>'), pStreetInfo(MAIL_INFO),
name_of_city(MAIL_INFO,_,NCITY), print('<name_of_city>'),
print(NCITY), print('</name_of_city>'), nl,
name_of_state(MAIL_INFO,_,NSTATE), print('<name_of_state>'),
print(NSTATE), print('</name_of_state>'), nl,
zip_code(MAIL_INFO,_,ZCODE), print('<zip_code>'), print(ZCODE),
print('</zip_code>'), nl, country(MAIL_INFO,COUNTRY_ID),
name(COUNTRY_ID,_,NCOUNTRY), print('<country><name>'),
print(NCOUNTRY), print('</name>'), nl,
exchange_rate(COUNTRY_ID,_,ERATE), print('<exchange_rate>'),
print(ERATE), print('<exchange_rate>'), nl,
currency(COUNTRY_ID,_,CURR), print('<currency>'), print(CURR),
print('</currency></country>'), print('</mailing_address>'), nl.

pContactInfoPublisher(ID) :- contact_information(ID,CINFO),
print('<contact_information>'), pMailingAddressPublisher(CINFO),
phone_number(CINFO,_,NUMBER), print('<phone_number>'), print(NUMBER),
print('</phone_number>'), nl, web_site(CINFO,_,URL), print('<web_site
>'), print(URL), print('</web_site >'), nl,
print('</contact_information>'), nl.

pStreetInfo_1([]).
pStreetInfo_1([H|T]) :- print('<street_address>'), print(H),
```

```
print('</street_address>'), nl, pStreetInfo_1(T).
pStreetInfo(ID) :- street_information(ID,X),
print('<street_information>'),
findall(Y,street_address(X,_,Y),LISTA), pStreetInfo_1(LISTA),
print('</street_information>').

pMailingAddress(ID) :- mailing_address(ID,MAIL_INFO),
print('<mailing_address>'), pStreetInfo(MAIL_INFO),
name_of_city(MAIL_INFO,_,NCITY), print('<name_of_city>'),
print(NCITY), print('</name_of_city>'), nl,
name_of_state(MAIL_INFO,_,NSTATE), print('<name_of_state>'),
print(NSTATE), print('</name_of_state>'), nl,
zip_code(MAIL_INFO,_,ZCODE), print('<zip_code>'), print(ZCODE),
print('</zip_code>'), nl, name_of_country(MAIL_INFO,_,NCOUNTRY),
print('<name_of_country>'), print(NCOUNTRY),
print('</name_of_country>'), nl, print('</mailing_address>'), nl.

pContactInfo(ID) :- contact_information(ID,CINFO),
print('<contact_information>'), pMailingAddress(CINFO),
phone_number(CINFO,_,NUMBER), print('<phone_number>'), print(NUMBER),
print('</phone_number>'), nl, email_address(CINFO,_,EMAIL),
print('<email_address>'), print(EMAIL), print('</email_address>'),
nl, print('</contact_information>'), nl.

pPublisher(ID) :- publisher(ID,PUB), name(PUB,_,NAME),
rint('<publisher>'), print('<name>'), print(NAME), print('</name>'),
nl, pContactInfoPublisher(PUB), print('</publisher>').

pName(ID) :-  name(ID,NAME_ID), print('<name>'),
first_name(NAME_ID,_,NAME), print('<first_name>'), print(NAME),
print('</first_name>'), nl, middle_name(NAME_ID,_,MNAME),
print('<middle_name>'), print(MNAME), print('</middle_name>'), nl,
last_name(NAME_ID,_,LNAME), print('<last_name>'), print(LNAME),
print('</last_name>'), nl, print('</name>'), nl.

pAuthors([]).
pAuthors([H|T]) :- print('<author>'), pName(H),
date_of_birth(H,_,DATE), print('<date_of_birth>'), print(DATE),
print('</date_of_birth>'), nl, biography(H,_,BIO),
print('<biography>'), print(BIO), print('</biography>'), nl,
pContactInfo(H), print('</author>'), nl, pAuthors(T).
```

```
pRelItens([]).
pRelItens([H|T]) :- item_id(H,_,ITEM), print('<related_item>'),
print('<item_id>'), print(ITEM), print('</item_id>'), nl,
print('</related_item>'), pRelItens(T).

dumpRelItens(ID) :- related_items(ID,REL_ITENS_ID),
findall(X,related_item(REL_ITENS_ID,X),REL_ITENS_LIST),
print('<related_items>'), nl, pRelItens(REL_ITENS_LIST), nl,
print('</related_items>').

dumpAuthors(ID) :- authors(ID,AUTHORS_ID),
findall(X,author(AUTHORS_ID,X),AUTHORS_LIST), print('<authors>'), nl,
pAuthors(AUTHORS_LIST), print('</authors>').

pItem(IID) :- id(ID,_,IID), print('<item id="'), print(IID),
print('">'), title(ID,_,TITLE), print('<title>'), print(TITLE),
print('</title>'), nl, dumpAuthors(ID),
date_of_release(ID,_,DT_RELEASE), print('<date_of_release>'),
print(DT_RELEASE), print('</date_of_release>'), nl, pPublisher(ID),
subject(ID,_,SUBJECT), print('<subject>'), print(SUBJECT),
print('/subject>'), nl, description(ID,_,DESC),
print('<description>'), print(DESC), print('</description>'), nl,
dumpRelItens(ID), pMedia(ID), pPricing(ID), pAttributes(ID),
print('</item>').
```

- **Q8** - Return the publisher of an item with id attribute value (I4).

  - **XQuery**

```
for $a in input()/catalog/*[@id="I4"]
return
   $a/publisher
```

  - **Prolog**

```
pConsultaOito :- id(ID,_,'I4'), pPublisher(ID).
pPublisher(ID) :- publisher(ID,PUB), name(PUB,_,NAME),
print('<publisher>'), print('<name>'), print(NAME), print('</name>'),
nl, pContactInfoPublisher(PUB), print('</publisher>').
```

```
pMailingAddressPublisher(ID) :- mailing_address(ID,MAIL_INFO),
print('<mailing_address>'), pStreetInfo(MAIL_INFO),
name_of_city(MAIL_INFO,_,NCITY), print('<name_of_city>'),
print(NCITY), print('</name_of_city>'), nl,
name_of_state(MAIL_INFO,_,NSTATE), print('<name_of_state>'),
print(NSTATE), print('</name_of_state>'), nl,
zip_code(MAIL_INFO,_,ZCODE), print('<zip_code>'), print(ZCODE),
print('</zip_code>'), nl, country(MAIL_INFO,COUNTRY_ID),
name(COUNTRY_ID,_,NCOUNTRY), print('<country><name>'),
print(NCOUNTRY), print('</name>'), nl,
exchange_rate(COUNTRY_ID,_,ERATE), print('<exchange_rate>'),
print(ERATE), print('<exchange_rate>'), nl,
currency(COUNTRY_ID,_,CURR), print('<currency>'), print(CURR),
print('</currency></country>'), print('</mailing_address>'), nl.

pContactInfoPublisher(ID) :- contact_information(ID,CINFO),
print('<contact_information>'), pMailingAddressPublisher(CINFO),
phone_number(CINFO,_,NUMBER), print('<phone_number>'), print(NUMBER),
print('</phone_number>'), nl, web_site(CINFO,_,URL),
print('<web_site>'), print(URL), print('</web_site >'), nl,
print('</contact_information>'), nl.

pStreetInfo_1([]).
pStreetInfo_1([H|T]) :- print('<street_address>'), print(H),
print('</street_address>'), nl, pStreetInfo_1(T).
pStreetInfo(ID) :- street_information(ID,X),
print('<street_information>'),
findall(Y,street_address(X,_,Y),LISTA), pStreetInfo_1(LISTA),
print('</street_information>').
```

- **Q9** - Return the ISBN of an item with id attribute value (I5).

  - **XQuery**

```
for $a in input()/catalog/:item
where $a/@id="I5"
return
    $a//ISBN/text()
```

  - **Prolog**

```
pConsultaNove :- id(_1,_,'I5'), attributes(_1,_2), isbn(_2,_,ISBN),
```

```
print(ISBN).
```

- **Q10 -** List the item titles ordered alphabetically by publisher name, with release date within a certain time period (from 1990-01-01 to 1995-01-01).

  - **XQuery**

```
for $a in input()/catalog/:item
where $a/date_of_release gt "1990-01-01" and
    $a/date_of_release lt "1995-01-01"
order by $a/publisher/name
return
    <Output>
        {$a/title}
        {$a/publisher}
    </Output>
```

  - **Prolog**

```
pConsultaDez :- findall(X,date_of_release(X,_,Y),LISTA),
maiores_com_id(LISTA,'1990-01-01',LISTA_MAIOR),
menores_com_id(LISTA_MAIOR,'1995-01-01',LISTA_MAIOR_MENOR),
ordena_C10(LISTA_MAIOR_MENOR,LISTA_ORDENADA), print('<Output>'),
pConsultaDez_1(LISTA_ORDENADA), print('</Output>').

pConsultaDez_1([]).
pConsultaDez_1([H|T]) :- title(H,_,TITLE), print('<title>'),
print(TITLE), print('</title>'), nl, pPublisher(H),
pConsultaDez_1(T).

maiores_com_id([],_,[]).
maiores_com_id([E|L],D,K) :- date_of_release(E,_,DT_E), (DT_E @< D),
maiores_com_id(L,D,K).
maiores_com_id([E|L],D,[E|K]) :- date_of_release(E,_,DT_E), (DT_E @>=
D) , maiores_com_id(L,D,K).

menores_com_id([],_,[]).
menores_com_id([E|L],D,K) :- date_of_release(E,_,DT_E), (DT_E @> D),
menores_com_id(L,D,K).
menores_com_id([E|L],D,[E|K]) :- date_of_release(E,_,DT_E), (DT_E @=<
D) , menores_com_id(L,D,K).
```

```
ordena_C10([],[]).
ordena_C10([X],[X]).
ordena_C10([X,Y|Z],S) :-
distribui_C10([X,Y|Z],A,B),
ordena_C10(A,As),
ordena_C10(B,Bs),
intercala_C10(As,Bs,S).

distribui_C10([],[],[]).
distribui_C10([X],[X],[]).
distribui_C10([X,Y|Z],[X|A],[Y|B]) :- distribui_C10(Z,A,B).

intercala_C10([],B,B).
intercala_C10(A,[],A).

intercala_C10([X|A],[Y|B],[X|C]) :- publisher(X,ID_PUB_X),
name(ID_PUB_X,_,NAME_PUB_X), publisher(Y,ID_PUB_Y),
name(ID_PUB_Y,_,NAME_PUB_Y), NAME_PUB_X @=< NAME_PUB_Y,
intercala_C10(A,[Y|B],C).

intercala_C10([X|A],[Y|B],[Y|C]) :- publisher(X,ID_PUB_X),
name(ID_PUB_X,_,NAME_PUB_X), publisher(Y,ID_PUB_Y),
name(ID_PUB_Y,_,NAME_PUB_Y), NAME_PUB_X @> NAME_PUB_Y,
intercala_C10([X|A],B,C).
```

- **Q11** - List the item titles in descending order by date of release with date of release within a certain time range (from 1990-01-01 to 1995-01-01).

  - **XQuery**

```
for $a in input()/catalog/:item
where $a/date_of_release gt "1990-01-01" and
    $a/date_of_release lt "1995-01-01"
order by $a/data_of_release descending
return
    <Output>
        {$a/title}
        {$a/date_of_release}
    </Output>
```

- **Prolog**

```
pConsultaOnze :- findall(X,date_of_release(X,_,Y),LISTA),
maiores_com_id(LISTA,'1990-01-01',LISTA_MAIOR),
menores_com_id(LISTA_MAIOR,'1995-01-01',LISTA_MAIOR_MENOR),
ordena_C11(LISTA_MAIOR_MENOR,LISTA_ORDENADA), print('<Output>')
pConsultaOnze_1(LISTA_ORDENADA), print('</Output>').

pConsultaOnze_1([]).
pConsultaOnze_1([H|T]) :- title(H,_,TITLE), print('<title>'),
print(TITLE), print('</title>'), nl, date_of_release(H,_,DT_RELEASE),
print('<date_of_release>'), print(DT_RELEASE),
print('</date_of_release>'), nl, pConsultaOnze_1(T).

maiores_com_id([],_,[]).
maiores_com_id([E|L],D,K) :- date_of_release(E,_,DT_E), (DT_E @< D),
maiores_com_id(L,D,K).
maiores_com_id([E|L],D,[E|K]) :- date_of_release(E,_,DT_E), (DT_E @>=
D) , maiores_com_id(L,D,K).

menores_com_id([],_,[]).
menores_com_id([E|L],D,K) :- date_of_release(E,_,DT_E), (DT_E @> D),
menores_com_id(L,D,K).
menores_com_id([E|L],D,[E|K]) :- date_of_release(E,_,DT_E), (DT_E @=<
D) , menores_com_id(L,D,K).

ordena_C11([],[]).
ordena_C11([X],[X]).
ordena_C11([X,Y|Z],S) :-
distribui_C11([X,Y|Z],A,B),
ordena_C11(A,As),
ordena_C11(B,Bs),
intercala_C11(As,Bs,S).

distribui_C11([],[],[]).
distribui_C11([X],[X],[]).
distribui_C11([X,Y|Z],[X|A],[Y|B]) :- distribui_C11(Z,A,B).

intercala_C11([],B,B).
intercala_C11(A,[],A).
```

```
intercala_C11([X|A],[Y|B],[X|C]) :- date_of_release(X,_,DT_X),
date_of_release(Y,_,DT_Y), DT_X @> DT_Y, intercala_C11(A,[Y|B],C).

intercala_C11([X|A],[Y|B],[Y|C]) :- date_of_release(X,_,DT_X),
date_of_release(Y,_,DT_Y), DT_X @=< DT_Y,  intercala_C11([X|A],B,C).
```

- **Q12** - Get the mailing address of the first author of certain item with id attribute value (I6).

  - **XQuery**

```
for $a in input()/catalog/:item[@id="I6"]
return
    <Output>
        {$a/authors/author[1]/contact_information/mailing_address}
    </Output>
```

  - **Prolog**

```
pConsultaDoze :- id(_1,_,'I6'), authors(_1, _2), author(_2, _3),
contact_information(_3, _4), print('<Output>'), nl,
pMailingAddress(_4), print('</Output>').

pMailingAddress(ID) :- mailing_address(ID,MAIL_INFO),
print('<mailing_address>'), pStreetInfo(MAIL_INFO),
name_of_city(MAIL_INFO,_,NCITY), print('<name_of_city>'),
print(NCITY), print('</name_of_city>'), nl,
name_of_state(MAIL_INFO,_,NSTATE), print('<name_of_state>'),
print(NSTATE), print('</name_of_state>'), nl,
zip_code(MAIL_INFO,_,ZCODE), print('<zip_code>'), print(ZCODE),
print('</zip_code>'), nl, name_of_country(MAIL_INFO,_,NCOUNTRY),
print('<name_of_country>'), print(NCOUNTRY),
print('</name_of_country>'), nl, print('</mailing_address>'), nl.


pStreetInfo(ID) :- street_information(ID,X),
print('<street_information>'),
findall(Y,street_address(X,_,Y),LISTA), pStreetInfo_1(LISTA),
print('</street_information>').
pStreetInfo_1([]).
pStreetInfo_1([H|T]) :- print('<street_address>'), print(H),
```

```
print('</street_address>'), nl, pStreetInfo_1(T).
```

- **Q14** - Return the names of publishers who publish books between a period (from 1990-01-01 to 1991-01-01) but do not have FAX number.

  - **XQuery**

```
for $a in input()/catalog/:item
where $a/date_of_release gt "1990-01-01" and
   $a/date_of_release lt "1991-01-01" and
   empty($a/publisher/contact_information/FAX_number)
return
   <Output>
      {$a/publisher/name}
   </Output>
```

  - **Prolog**

```
pConsultaQuatorze :- findall(X,date_of_release(X,_,Y),LISTA),
maiores_com_id(LISTA,'1990-01-01',LISTA_MAIOR),
menores_com_id(LISTA_MAIOR,'1991-01-01',LISTA_MAIOR_MENOR),
print('<Output>'), nl, pConsultaQuatorze_1(LISTA_MAIOR_MENOR),
print('</Output>').

pConsultaQuatorze_1([]).

pConsultaQuatorze_1([H|T]) :- publisher(H,_1),
contact_information(_1,_2), not(fax_number(_2,_,FAX_NUMBER)),
name(_1,_,PUBLISHER_NAME), print('<publisher'>), print('<name>'),
print(PUBLISHER_NAME), print('</name>'), print('</publisher'>), nl,
pConsultaQuatorze_1(T).

pConsultaQuatorze_1([H|T]) :- publisher(H,_1),
contact_information(_1,_2), fax_number(_2,_,FAX_NUMBER),
pConsultaQuatorze_1(T).

maiores_com_id([],_,[]).
maiores_com_id([E|L],D,K) :- date_of_release(E,_,DT_E), (DT_E @< D),
maiores_com_id(L,D,K).
maiores_com_id([E|L],D,[E|K]) :- date_of_release(E,_,DT_E), (DT_E @>=
D) , maiores_com_id(L,D,K).
```

```
menores_com_id([],_,[]).
menores_com_id([E|L],D,K) :- date_of_release(E,_,DT_E), (DT_E @> D),
menores_com_id(L,D,K).
menores_com_id([E|L],D,[E|K]) :- date_of_release(E,_,DT_E), (DT_E @=<
D) , menores_com_id(L,D,K).
```

- **Q17** - Return the ids of items whose descriptions contain a certain word (``hockey").
  - **XQuery**

```
for $a in input()/catalog/:item
where contains ($a/description, "hockey")
return
     <Output>
         {$a/@id}
     </Output>
```

  - **Prolog**

```
pConsultaDezessete :- findall(X,id(X,_,Y),LISTA), print('<Output>'),
nl, pConsultaDezessete_1(LISTA), print('</Output>')..

pConsultaDezessete_1([]).
pConsultaDezessete_1([H|T]) :- description(H,_,DESC),
sub_atom(DESC,_,_,_,'hockey'), id(H,_,ID), print('<id>'), print(ID),
print('</id>'), nl, pConsultaDezessete_1(T).

pConsultaDezessete_1([H|T]) :- description(H,_,DESC),
not(sub_atom(DESC,_,_,_,'hockey')), pConsultaDezessete_1(T).
```

- **Q19** - Retrieve the item titles related by certain item with id attribute value (I7).
  - **XQuery**

```
for $item in input()/catalog/:item[@id="I7"],
    $related in input()/catalog/:item
where $item/related_items/related_item/item_id = $related/@id
return
    <Output>
        {$related/title}
```

```
    </Output>
```

- **Prolog**

```
pConsultaDezenove :- id(X,_,'I7'), related_items(X,Y),
findall(W,(related_item(Y,Z),item_id(Z,_,W)),LISTA),
print('<Output>'), nl,  pConsultaDezenove_1(LISTA),
print('</Output>').

pConsultaDezenove_1([]).
pConsultaDezenove_1([H|T]) :- id(X,_,H), title(X,_,TITLE),
print(TITLE), nl, pConsultaDezenove_1(T).
```

- **Q20** - Retrieve the item title whose size (length*width*height) is bigger than certain number (500000).

- **XQuery**

```
for $size in input()/catalog/:item/attributes/size_of_book
where $size/length*$size/width*$size/height > 500000
return
    <Output>
        {$size/../../title}
    </Output>
```

- **Prolog**

```
pConsultaVinte :- findall(X,size_of_book(X,Y),LISTA),
pConsultaVinte_1(LISTA).

pConsultaVinte_1([]).

pConsultaVinte_1([H|T]):- size_of_book(H,SIZE_ID),
length(SIZE_ID,LIXO,LENGTH_TEMP), atom_number(LENGTH_TEMP,LENGTH),
width(SIZE_ID,LIXO2,WIDTH_TEMP),atom_number(WIDTH_TEMP,WIDTH),
height(SIZE_ID,LIXO3,HEIGHT_TEMP), atom_number(HEIGHT_TEMP,HEIGHT),
RESULT is LENGTH * WIDTH * HEIGHT, not(RESULT > 500000),
pConsultaVinte_1(T).

pConsultaVinte_1([H|T]):- size_of_book(H,SIZE_ID),
length(SIZE_ID,LIXO,LENGTH_TEMP), atom_number(LENGTH_TEMP,LENGTH),
width(SIZE_ID,LIXO2,WIDTH_TEMP),atom_number(WIDTH_TEMP,WIDTH),
```

```
height(SIZE_ID,LIXO3,HEIGHT_TEMP), atom_number(HEIGHT_TEMP,HEIGHT),
RESULT is LENGTH * WIDTH * HEIGHT, RESULT > 500000,
attributes(ITEM_ID,H), title(ITEM_ID,_,TITLE), print('<title>'),
print(TITLE), print('</title>'), nl, pConsultaVinte_1(T).
```

1.2 SINGLE DOCUMENT TEXT CENTRIC XQUERY QUERIES

In this section we present the single document/text centric XQuery queries and the corresponding Prolog queries.

- **Q1** - Return the entry that has matching headword ("the").

  - **XQuery**

```
for $ent in input()/dictionary/e
where $ent/hwg/hw="lethe"
return
   $ent
```

  - **Prolog**

```
pConsulta01 :- hw(_1,_,'lethe'), hwg(_2,_1), pE(_2).

pE(ID) :- id(ID,_,ATT_ID), findall(X,hwg(ID,X),L1),
findall(Y,vfl(ID,Y),L2), findall(Z,et(ID,Z),L3),
findall(W,ss(ID,W),L4), print('<e id="'), print(ATT_ID), print('">'),
nl, pE_01(L1), pE_02(L2), pE_03(L3), pE_04(L4), print('</e>'), nl.
pE_01([]).
pE_01([H|T]) :- pHWG(H), pE_01(T).
pE_02([]).
pE_02([H|T]) :- pVFL(H), pE_02(T).
pE_03([]).
pE_03([H|T]) :- pET(H), pE_03(T).
pE_04([]).
pE_04([H|T]) :- pSS(H), pE_04(T).

pSS(ID) :- findall(X,s(ID,X),L1), print('<ss>'), nl,  pSS_01(L1),
print('</ss>'), nl.
pSS_01([H|T]) :- pS(H), pSS_01(T).
pSS_01([]).
```

```
pS(ID) :- print('<s>'), nl, def(ID,_,DEF), print('<def>'),
print(DEF), print('</def>'), nl, pQP(ID), print('</s>'), nl, !.

pS(ID) :- print('<s>'), nl, def(ID,DEF), findall(B-
(mixedElement)/C,xml/mixedElement(DEF,B,C),LISTA1), findall(B-
(cr)/C,cr(DEF,B,C),LISTA2), append(LISTA1,LISTA2,LISTA_FIM),
keysort(LISTA_FIM,ORDERED), print('<def>'), nl,
printListAfterKeySort(ORDERED), print('</def>'), nl, pQP(ID),
print('</s>'), nl.

pQP(ID) :- findall(X,qp(ID,X),LISTA), pQP_01(LISTA).
pQP_01([]).
pQP_01([H|T]) :- print('<qp>'), nl, pQ(H), print('</qp>'), nl,
pQP_01(T).

pQ(ID) :- findall(X,q(ID,X),LISTA), pQ_01(LISTA).
pQ_01([]).
pQ_01([H|T]) :- print('<q>'), nl, findall(_1,qd(H,_,_1),LISTA1),
pLISTA(LISTA1,qd), findall(_2,a(H,_,_2),LISTA2), pLISTA(LISTA2,a),
findall(_3,w(H,_,_3),LISTA3), pLISTA(LISTA3,w),
findall(_4,bib(H,_,_4),LISTA4), pLISTA(LISTA4,bib),
findall(_5,loc(H,_,_5),LISTA5), pLISTA(LISTA5,loc), nl, pQT(H),
print('</q>'), pQ_01(T).

pQT(ID) :- qt(ID,_,X), print('<qt>'), print(X), print('</qt>'), nl.
pQT(ID) :- findall(X,qt(ID,X),LISTA), pQT_01(LISTA).
pQT_01([]).
pQT_01([H|T]) :- print('<qt>'), nl, findall(B-
(cr)/C,cr(H,B,C),LISTA), findall(B-(i)/C,i(H,B,C),LISTA2), findall(B-
(b)/C,b(H,B,C),LISTA3), findall(B-
(mixedElement)/C,xml/mixedElement(H,B,C),LISTA4),
append(LISTA,LISTA2,TMP), append(TMP,LISTA3,TMP2),
append(TMP2,LISTA4,TMP3), keysort(TMP3,ORDERED),
printListAfterKeySort(ORDERED), nl, print('</qt>'), nl, pQT_01(T).

pHWG(ID) :- findall(B-(hw)/C,hw(ID,B,C),LISTA), findall(B-
(pr)/C,pr(ID,B,C),LISTA2), findall(B-(pos)/C,pos(ID,B,C),LISTA3),
append(LISTA,LISTA2,TMP), append(TMP,LISTA3,TMP2),
keysort(TMP2,ORDERED), print('<hwg>'), nl,
printListAfterKeySort(ORDERED), print('</hwg>'), nl.
```

```
pET(ID) :- findall(X,cr(ID,_,X),L1), print('<et>'), nl,
pLISTA(L1,'cr'), print('</et>'), nl.

pVFL(ID) :- findall(B-(vf)/C,vf(ID,B,C),LISTA), findall(B-
(vd)/C,vd(ID,B,C),LISTA2), append(LISTA,LISTA2,FINAL),
keysort(FINAL,ORDERED), print('<vfl>'), nl,
printListAfterKeySort(ORDERED), print('</vfl>'), nl.
```

- **Q2** - Find the headword of the entry which has matching quotation year (1900).

  - **XQuery**

```
for $ent in input()/dictionary/e
where $ent/ss/s/qp/q/qd="1900"
return
   $ent/hwg/hw
```

  - **Prolog**

```
pConsulta02 :- findall(X,qd(X,_,'1900'),LISTA), pConsulta02_1(LISTA).
pConsulta02_1([]).
pConsulta02_1([H|T]) :- q(_1,H), qp(_2,_1), s(_3,_2), ss(_4,_3),
hwg(_4,_5), findall(X,hw(_5,_,X),LISTA_2), pConsulta02_2(LISTA_2),
pConsulta02_1(T).
pConsulta02_2([]).
pConsulta02_2([H|T]) :- print('<hw>'), print(H), print('</hw>'), nl,
pConsulta02_2(T).
```

- **Q3** - Group entries by quotation location in a certain quotation year (1900) and calculate the total number entries in each group.

  - **XQuery**

```
for $a in distinct-values
    (input()/dictionary/e/ss/s/qp/q[qd="1900"]/loc)
let $b := input()/dictionary/e/ss/s/qp/q[loc=$a]
return
    <Output>
        <Location>{$a/text()}</Location>
```

```
        <NumberOfEntries>{count($b)}</NumberOfEntries>
    </Output>
```

- **Prolog**

```
pConsulta03_01([]).
pConsulta03_01([H|T]) :- findall(X,loc(X,_,H),LISTA2),
print('<Location>'), print(H), print('</Location>'), nl,
print('<NumberOfEntries>'), length(LISTA2,N), print(N),
print('</NumberOfEntries>'), nl, pConsulta03_01(T).
pConsulta03 :- print('<Output>'), nl,
setof(Z,X^Y^W^(loc(X,Y,Z),qd(X,W,'1900')),LISTA),
pConsulta03_01(LISTA), print('</Output>'), nl.
```

- **Q4** - List the headword of the previous entry of a matching headword (``hearse").

- **XQuery**

```
let $ent := input()/dictionary/e[hwg/hw="hearse"]
for $prevEnt in input()/dictionary/e[hwg/hw << $ent]
[position() = last()]
return
   <Output>
      <CurrentEntry>{$ent/hwg/hw/text()}</CurrentEntry>
      <PreviousEntry>{$prevEnt/hwg/hw/text()}</PreviousEntry>
   </Output>
```

- **Prolog**

```
pConsulta04 :- findall(Y,hw(X,_,Y),LISTA), pConsulta04_01(LISTA,'No
Previous').
pConsulta04_01([],_).
pConsulta04_01([H|T],K) :- H = 'hearse', print('<PreviousEntry>'),
print(H), print('</PreviousEntry >'), nl, print('<CurrentEntry>'),
print(K), print('<CurrentEntry>').
pConsulta04_01([H|T],K) :- not(H = 'hearse'), pConsulta04_01(T,H).
```

- **Q5** - Return the first sense of a matching headword (“`hearse`”).

- **XQuery**

```
for $a in input()/dictionary/e
where $a/hwg/hw="hearse"
return
   $a/ss/s[1]
```

- **Prolog**

```
pConsulta05 :- hw(_1,_,'hearse'), hwg(_2,_1), ss(_2,_3), s(_3,_4),
pS(_4).

pS(ID) :- print('<s>'), nl, def(ID,_,DEF), print('<def>'),
print(DEF), print('</def>'), nl, pQP(ID), print('</s>'), nl, !.

pS(ID) :- print('<s>'), nl, def(ID,DEF), findall(B-
(mixedElement)/C,xml/mixedElement(DEF,B,C),LISTA1), findall(B-
(cr)/C,cr(DEF,B,C),LISTA2), append(LISTA1,LISTA2,LISTA_FIM),
keysort(LISTA_FIM,ORDERED), print('<def>'), nl,
printListAfterKeySort(ORDERED), print('</def>'), nl, pQP(ID),
print('</s>'), nl.


pQP(ID) :- findall(X,qp(ID,X),LISTA), pQP_01(LISTA).
pQP_01([]).
pQP_01([H|T]) :- print('<qp>'), nl, pQ(H), print('</qp>'), nl,
pQP_01(T).

pQ(ID) :- findall(X,q(ID,X),LISTA), pQ_01(LISTA).
pQ_01([]).
pQ_01([H|T]) :- print('<q>'), nl, findall(_1,qd(H,_,_1),LISTA1),
pLISTA(LISTA1,qd), findall(_2,a(H,_,_2),LISTA2), pLISTA(LISTA2,a),
findall(_3,w(H,_,_3),LISTA3), pLISTA(LISTA3,w),
findall(_4,bib(H,_,_4),LISTA4), pLISTA(LISTA4,bib),
findall(_5,loc(H,_,_5),LISTA5), pLISTA(LISTA5,loc), nl, pQT(H),
print('</q>'), pQ_01(T).

pQT(ID) :- qt(ID,_,X), print('<qt>'), print(X), print('</qt>'), nl.
pQT(ID) :- findall(X,qt(ID,X),LISTA), pQT_01(LISTA).
pQT_01([]).
pQT_01([H|T]) :- print('<qt>'), nl, findall(B-
(cr)/C,cr(H,B,C),LISTA), findall(B-(i)/C,i(H,B,C),LISTA2), findall(B-
(b)/C,b(H,B,C),LISTA3), findall(B-
(mixedElement)/C,xml/mixedElement(H,B,C),LISTA4),
```

```
append(LISTA,LISTA2,TMP), append(TMP,LISTA3,TMP2),
append(TMP2,LISTA4,TMP3), keysort(TMP3,ORDERED),
printListAfterKeySort(ORDERED), nl, print('</qt>'), nl, pQT_01(T).
```

- **Q6** - Return the words where some quotations were quoted in a certain year (1900).

  - **XQuery**

```
for $word in input()/dictionary/e
where some $item in $word/ss/s/qp/q
satisfies $item/qd eq "1900"
return
   $word
```

  - **Prolog**

```
pConsulta06 :-
setof(ID,A^B^C^D^E^(qd(B,A,'1900'),q(C,B),qp(D,C),s(E,D),ss(ID,E)),LI
STA), pConsulta06_01(LISTA).
pConsulta06_01([]).
pConsulta06_01([H|T]) :- pE(H), pConsulta06_01(T).

pE(ID) :- id(ID,_,ATT_ID), findall(X,hwg(ID,X),L1),
findall(Y,vfl(ID,Y),L2), findall(Z,et(ID,Z),L3),
findall(W,ss(ID,W),L4), print('<e id="'), print(ATT_ID), print('">'),
nl, pE_01(L1), pE_02(L2), pE_03(L3), pE_04(L4), print('</e>'), nl.
pE_01([]).
pE_01([H|T]) :- pHWG(H), pE_01(T).
pE_02([]).
pE_02([H|T]) :- pVFL(H), pE_02(T).
pE_03([]).
pE_03([H|T]) :- pET(H), pE_03(T).
pE_04([]).
pE_04([H|T]) :- pSS(H), pE_04(T).

pSS(ID) :- findall(X,s(ID,X),L1), print('<ss>'), nl,  pSS_01(L1),
print('</ss>'), nl.
pSS_01([H|T]) :- pS(H), pSS_01(T).
pSS_01([]).

pS(ID) :- print('<s>'), nl, def(ID,_,DEF), print('<def>'),
```

```
print(DEF), print('</def>'), nl, pQP(ID), print('</s>'), nl, !.

pS(ID) :- print('<s>'), nl, def(ID,DEF), findall(B-
(mixedElement)/C,xml/mixedElement(DEF,B,C),LISTA1), findall(B-
(cr)/C,cr(DEF,B,C),LISTA2), append(LISTA1,LISTA2,LISTA_FIM),
keysort(LISTA_FIM,ORDERED), print('<def>'), nl,
printListAfterKeySort(ORDERED), print('</def>'), nl, pQP(ID),
print('</s>'), nl.

pQP(ID) :- findall(X,qp(ID,X),LISTA), pQP_01(LISTA).
pQP_01([]).
pQP_01([H|T]) :- print('<qp>'), nl, pQ(H), print('</qp>'), nl,
pQP_01(T).

pQ(ID) :- findall(X,q(ID,X),LISTA), pQ_01(LISTA).
pQ_01([]).
pQ_01([H|T]) :- print('<q>'), nl, findall(_1,qd(H,_,_1),LISTA1),
pLISTA(LISTA1,qd), findall(_2,a(H,_,_2),LISTA2), pLISTA(LISTA2,a),
findall(_3,w(H,_,_3),LISTA3), pLISTA(LISTA3,w),
findall(_4,bib(H,_,_4),LISTA4), pLISTA(LISTA4,bib),
findall(_5,loc(H,_,_5),LISTA5), pLISTA(LISTA5,loc), nl, pQT(H),
print('</q>'), pQ_01(T).

pQT(ID) :- qt(ID,_,X), print('<qt>'), print(X), print('</qt>'), nl.
pQT(ID) :- findall(X,qt(ID,X),LISTA), pQT_01(LISTA).
pQT_01([]).
pQT_01([H|T]) :- print('<qt>'), nl, findall(B-
(cr)/C,cr(H,B,C),LISTA), findall(B-(i)/C,i(H,B,C),LISTA2), findall(B-
(b)/C,b(H,B,C),LISTA3), findall(B-
(mixedElement)/C,xml/mixedElement(H,B,C),LISTA4),
append(LISTA,LISTA2,TMP), append(TMP,LISTA3,TMP2),
append(TMP2,LISTA4,TMP3), keysort(TMP3,ORDERED),
printListAfterKeySort(ORDERED), nl, print('</qt>'), nl, pQT_01(T).

pHWG(ID) :- findall(B-(hw)/C,hw(ID,B,C),LISTA), findall(B-
(pr)/C,pr(ID,B,C),LISTA2), findall(B-(pos)/C,pos(ID,B,C),LISTA3),
append(LISTA,LISTA2,TMP), append(TMP,LISTA3,TMP2),
keysort(TMP2,ORDERED), print('<hwg>'), nl,
printListAfterKeySort(ORDERED), print('</hwg>'), nl.

pET(ID) :- findall(X,cr(ID,_,X),L1), print('<et>'), nl,
```

```
pLISTA(L1,'cr'), print('</et>'), nl.

pVFL(ID) :- findall(B-(vf)/C,vf(ID,B,C),LISTA), findall(B-
(vd)/C,vd(ID,B,C),LISTA2), append(LISTA,LISTA2,FINAL),
keysort(FINAL,ORDERED), print('<vfl>'), nl,
printListAfterKeySort(ORDERED), print('</vfl>'), nl.
```

- **Q7** - Return the words where all quotations were quoted in a certain year (1900).

  - **XQuery**

```
for $word in input()/dictionary/e
where every $item in $word/ss/s/qp/q
    satisfies $item/qd eq "1900"
return
    $wor
```

  - **Prolog**

```
pConsulta07 :- findall(Y,e(X,Y),LISTA), pConsulta07_01(LISTA).

pConsulta07_01([]).
pConsulta07_01([H|T]) :- findall(X,(ss(H,_1), s(_1,_2), qp(_2,_3),
q(_3,_4), qd(_4,_,X)),LISTA), pConsulta07_02(LISTA,''),
pConsulta07_01(T).

pConsulta07_02([],K) :- qd(_1,K), q(_2,_1), qp(_3,_2), s(_4,_3),
ss(E,_4), pE(E).
pConsulta07_02([H|T],K) :-  not(H = '1900'), !.
pConsulta07_02([H|T],K) :- pConsulta07_02(T).

pE(ID) :- id(ID,_,ATT_ID), findall(X,hwg(ID,X),L1),
findall(Y,vfl(ID,Y),L2), findall(Z,et(ID,Z),L3),
findall(W,ss(ID,W),L4), print('<e id="'), print(ATT_ID), print('">'),
nl, pE_01(L1), pE_02(L2), pE_03(L3), pE_04(L4), print('</e>'), nl.
pE_01([]).
pE_01([H|T]) :- pHWG(H), pE_01(T).
pE_02([]).
pE_02([H|T]) :- pVFL(H), pE_02(T).
pE_03([]).
pE_03([H|T]) :- pET(H), pE_03(T).
```

```
pE_04([]).
pE_04([H|T]) :- pSS(H), pE_04(T).

pSS(ID) :- findall(X,s(ID,X),L1), print('<ss>'), nl,  pSS_01(L1),
print('</ss>'), nl.
pSS_01([H|T]) :- pS(H), pSS_01(T).
pSS_01([]).

pS(ID) :- print('<s>'), nl, def(ID,_,DEF), print('<def>'),
print(DEF), print('</def>'), nl, pQP(ID), print('</s>'), nl, !.

pS(ID) :- print('<s>'), nl, def(ID,DEF), findall(B-
(mixedElement)/C,xml/mixedElement(DEF,B,C),LISTA1), findall(B-
(cr)/C,cr(DEF,B,C),LISTA2), append(LISTA1,LISTA2,LISTA_FIM),
keysort(LISTA_FIM,ORDERED), print('<def>'), nl,
printListAfterKeySort(ORDERED), print('</def>'), nl, pQP(ID),
print('</s>'), nl.

pQP(ID) :- findall(X,qp(ID,X),LISTA), pQP_01(LISTA).
pQP_01([]).
pQP_01([H|T]) :- print('<qp>'), nl, pQ(H), print('</qp>'), nl,
pQP_01(T).

pQ(ID) :- findall(X,q(ID,X),LISTA), pQ_01(LISTA).
pQ_01([]).
pQ_01([H|T]) :- print('<q>'), nl, findall(_1,qd(H,_,_1),LISTA1),
pLISTA(LISTA1,qd), findall(_2,a(H,_,_2),LISTA2), pLISTA(LISTA2,a),
findall(_3,w(H,_,_3),LISTA3), pLISTA(LISTA3,w),
findall(_4,bib(H,_,_4),LISTA4), pLISTA(LISTA4,bib),
findall(_5,loc(H,_,_5),LISTA5), pLISTA(LISTA5,loc), nl, pQT(H),
print('</q>'), pQ_01(T).

pQT(ID) :- qt(ID,_,X), print('<qt>'), print(X), print('</qt>'), nl.
pQT(ID) :- findall(X,qt(ID,X),LISTA), pQT_01(LISTA).
pQT_01([]).
pQT_01([H|T]) :- print('<qt>'), nl, findall(B-
(cr)/C,cr(H,B,C),LISTA), findall(B-(i)/C,i(H,B,C),LISTA2), findall(B-
(b)/C,b(H,B,C),LISTA3), findall(B-
(mixedElement)/C,xml/mixedElement(H,B,C),LISTA4),
append(LISTA,LISTA2,TMP), append(TMP,LISTA3,TMP2),
append(TMP2,LISTA4,TMP3), keysort(TMP3,ORDERED),
```

```
printListAfterKeySort(ORDERED), nl, print('</qt>'), nl, pQT_01(T).

pHWG(ID) :- findall(B-(hw)/C,hw(ID,B,C),LISTA), findall(B-
(pr)/C,pr(ID,B,C),LISTA2), findall(B-(pos)/C,pos(ID,B,C),LISTA3),
append(LISTA,LISTA2,TMP), append(TMP,LISTA3,TMP2),
keysort(TMP2,ORDERED), print('<hwg>'), nl,
printListAfterKeySort(ORDERED), print('</hwg>'), nl.

pET(ID) :- findall(X,cr(ID,_,X),L1), print('<et>'), nl,
pLISTA(L1,'cr'), print('</et>'), nl.

pVFL(ID) :- findall(B-(vf)/C,vf(ID,B,C),LISTA), findall(B-
(vd)/C,vd(ID,B,C),LISTA2), append(LISTA,LISTA2,FINAL),
keysort(FINAL,ORDERED), print('<vfl>'), nl,
printListAfterKeySort(ORDERED), print('</vfl>'), nl.
```

- **Q8** - Return Quotation Text (one element name unknown) of a word ("`shipboard`").

  - **XQuery**

```
for $ent in input()/dictionary/e
where $ent/*/hw = "shipboard"
return
    $ent/ss/s/qp/*/qt
```

  - **Prolog**

```
pConsulta08 :-
findall(Z,(e(X,Y),hwg(Y,Z),hw(Z,_,'shipboard')),LISTA),
pConsulta08_01(LISTA).
pConsulta08_01([]).
pConsulta08_01([H|T]) :- hwg(_1,H), ss(_1,_2),
findall(X,s(_2,X),LISTA), pConsulta08_02(LISTA), pConsulta08_01(T).
pConsulta08_02([]).
pConsulta08_02([H|T]) :- qp(H,_1), findall(X,q(_1,X),LISTA),
pConsulta08_03(LISTA), pConsulta08_02(T).
pConsulta08_03([]).
pConsulta08_03([H|T]) :- pQT(H), pConsulta08_03(T).

pQT(ID) :- qt(ID,_,X), print('<qt>'), print(X), print('</qt>'), nl.
```

```
pQT(ID) :- findall(X,qt(ID,X),LISTA), pQT_01(LISTA).
pQT_01([]).
pQT_01([H|T]) :- print('<qt>'), nl, findall(B-
(cr)/C,cr(H,B,C),LISTA), findall(B-(i)/C,i(H,B,C),LISTA2), findall(B-
(b)/C,b(H,B,C),LISTA3), findall(B-
(mixedElement)/C,xml/mixedElement(H,B,C),LISTA4),
append(LISTA,LISTA2,TMP), append(TMP,LISTA3,TMP2),
append(TMP2,LISTA4,TMP3), keysort(TMP3,ORDERED),
printListAfterKeySort(ORDERED), nl, print('</qt>'), nl, pQT_01(T).
```

- **Q9** - Return Quotation Text (several consecutive element names unknown) of a word (``and").

  - **XQuery**

```
for $ent in input()/dictionary/e
where $ent//hw = "or"
return
   $ent//qt
```

  - **Prolog**

```
pConsulta09 :- findall(X,hw(X,_,'shipboard'),LISTA),
pConsulta09_01(LISTA).
pConsulta09_01([]).
pConsulta09_01([H|T]) :- hwg(_1,H), ss(_1,_2),
findall(X,s(_2,X),LISTA), pConsulta09_02(LISTA), pConsulta09_01(T).
pConsulta09_02([]).
pConsulta09_02([H|T]) :- qp(H,_1), findall(X,q(_1,X),LISTA),
pConsulta09_03(LISTA), pConsulta09_02(T).
pConsulta09_03([]).
pConsulta09_03([H|T]) :- pQT(H), pConsulta09_03(T).

pQT(ID) :- qt(ID,_,X), print('<qt>'), print(X), print('</qt>'), nl.
pQT(ID) :- findall(X,qt(ID,X),LISTA), pQT_01(LISTA).
pQT_01([]).
pQT_01([H|T]) :- print('<qt>'), nl, findall(B-
(cr)/C,cr(H,B,C),LISTA), findall(B-(i)/C,i(H,B,C),LISTA2), findall(B-
(b)/C,b(H,B,C),LISTA3), findall(B-
(mixedElement)/C,xml/mixedElement(H,B,C),LISTA4),
append(LISTA,LISTA2,TMP), append(TMP,LISTA3,TMP2),
```

```
append(TMP2,LISTA4,TMP3), keysort(TMP3,ORDERED),
printListAfterKeySort(ORDERED), nl, print('</qt>'), nl, pQT_01(T).
```

- **Q10** - List the words and their pronunciation, alphabetically, quoted in a certain year (1900).

  - **XQuery**

```
for $a in input()/dictionary/e
where $a/ss/s/qp/q/qd = "1900"
order by $a/hwg/hw
return
   <Output>
      {$a/hwg/hw}
      {$a/hwg/pr}
   </Output>
```

  - **Prolog**

```
pConsulta10 :- findall(G-
H,(qd(A,_,'1900'),q(B,A),qp(C,B),s(D,C),ss(E,D),hwg(E,F),hw(F,_,G),pr
(F,_,H)),LISTA),msort(LISTA,LISTA2), pConsulta10_01(LISTA2).
pConsulta10_01([]).
pConsulta10_01([H1-H2|T]) :- print('<Output>'), nl, print('<hw>'),
print(H1), print('</hw>'), nl, print('<pr>'), print(H2),
print('</pr>'), nl, print('</Output>'), nl, pConsulta10_01(T).
```

- **Q11** - List the quotation locations and quotation dates, sorted by date, for a word (``word").

  - **XQuery**

```
for $a in input()/dictionary/e
    [hwg/hw="the"]/ss/s/qp/q
order by $a/qd
return
    <Output>
        {$a/a}
        {$a/qd}
    </Output>
```

- **Prolog**

```
pConsulta11 :- findall(QD-
V,(hw(X,_,'shipboard'),hwg(Y,X),ss(Y,Z),s(Z,W),qp(W,U),q(U,V),qd(V,_,
QD)),LISTA), keysort(LISTA,ORDERED), pConsulta11_01(ORDERED).

pConsulta11_01([]).

pConsulta11_01([H-N|T]) :- a(N,_,A), print('<output>'), nl,
print('<a>'), print(A), print('</a>'), nl, print('<qd>'), print(H),
print('</qd>'), nl, , print('</output>'), nl, pConsulta11_01(T), !.

pConsulta11_01([H-N|T]) :- print('<output>'), nl, print('<qd>'),
print(H), print('</qd>'), nl, print('</output>'), nl,
pConsulta11_01(T).
```

- **Q12** - Retrieve the senses of a word ("`valorous`").

- **XQuery**

```
for $a in input()/dictionary/e
where $a/hwg/hw="valorous"
return
   <Entry>
      {$a/ss}
   </Entry>
```

- **Prolog**

```
pConsulta12 :- findall(X,hw(X,_,'valorous'),LISTA),
pConsulta12_01(LISTA).
pConsulta12_01([]).
pConsulta12_01([H|T]) :- print('<Entry>'), nl, hwg(ID,H), ss(ID,_1),
pSS(_1), print('</Entry>'), nl.

pSS(ID) :- findall(X,s(ID,X),L1), print('<ss>'), nl,  pSS_01(L1),
print('</ss>'), nl.
pSS_01([H|T]) :- pS(H), pSS_01(T).
pSS_01([]).
```

```
pS(ID) :- print('<s>'), nl, def(ID,_,DEF), print('<def>'),
print(DEF), print('</def>'), nl, pQP(ID), print('</s>'), nl, !.

pS(ID) :- print('<s>'), nl, def(ID,DEF), findall(B-
(mixedElement)/C,xml/mixedElement(DEF,B,C),LISTA1), findall(B-
(cr)/C,cr(DEF,B,C),LISTA2), append(LISTA1,LISTA2,LISTA_FIM),
keysort(LISTA_FIM,ORDERED), print('<def>'), nl,
printListAfterKeySort(ORDERED), print('</def>'), nl, pQP(ID),
print('</s>'), nl.

pQP(ID) :- findall(X,qp(ID,X),LISTA), pQP_01(LISTA).
pQP_01([]).
pQP_01([H|T]) :- print('<qp>'), nl, pQ(H), print('</qp>'), nl,
pQP_01(T).

pQ(ID) :- findall(X,q(ID,X),LISTA), pQ_01(LISTA).
pQ_01([]).
pQ_01([H|T]) :- print('<q>'), nl, findall(_1,qd(H,_,_1),LISTA1),
pLISTA(LISTA1,qd), findall(_2,a(H,_,_2),LISTA2), pLISTA(LISTA2,a),
findall(_3,w(H,_,_3),LISTA3), pLISTA(LISTA3,w),
findall(_4,bib(H,_,_4),LISTA4), pLISTA(LISTA4,bib),
findall(_5,loc(H,_,_5),LISTA5), pLISTA(LISTA5,loc), nl, pQT(H),
print('</q>'), pQ_01(T).

pQT(ID) :- qt(ID,_,X), print('<qt>'), print(X), print('</qt>'), nl.
pQT(ID) :- findall(X,qt(ID,X),LISTA), pQT_01(LISTA).
pQT_01([]).

pQT_01([H|T]) :- print('<qt>'), nl, findall(B-
(cr)/C,cr(H,B,C),LISTA), findall(B-(i)/C,i(H,B,C),LISTA2), findall(B-
(b)/C,b(H,B,C),LISTA3), findall(B-
(mixedElement)/C,xml/mixedElement(H,B,C),LISTA4),
append(LISTA,LISTA2,TMP), append(TMP,LISTA3,TMP2),
append(TMP2,LISTA4,TMP3), keysort(TMP3,ORDERED),
printListAfterKeySort(ORDERED), nl, print('</qt>'), nl, pQT_01(T).
```

- **Q13** - Construct a brief information on a word ("`valorous`"), including: headword, pronunciation, part_of_speech, first etymology and first sense definition.

  - **XQuery**

```
for $a in input()/dictionary/e
where $a/hwg/hw="valorous"
return
    <Output>
        {$a/hwg/hw}
        {$a/hwg/pr}
        {$a/hwg/pos}
        {$a/etymology/cr[1]}
        {$a/ss/s[1]/def}
    </Output>
```

- **Prolog**

```
pConsulta13 :- print('<Output>'), nl, hw(_1,_,'valorous'),
pr(_1,_,PR), pos(_1,_,POS), print('<hw>'), print('valorous'),
print('</hw>'), nl, print('<pr>'), print(PR), print('</pr>'), nl,
print('<pos>'), print(POS), print('</pos>'), nl, hwg(_2,_1),
ss(_2,_4), s(_4,_5), def(_5,_,DEF), print('<def>'), print(DEF),
print('</def>'), nl, print('</Output>').
```

- **Q14** - List the ids of entries that do not have variant form lists and etymologies.

- **XQuery**

```
for $a in input()/dictionary/e
where empty($a/vfl) and empty($a/et)
return
   <NoVFLnET>
      {$a/@id}
   </NoVFLnET>
```

- **Prolog**

```
pConsulta14 :- findall(Y,e(X,Y),LISTA), pConsulta14_01(LISTA).
pConsulta14_01([]).
pConsulta14_01([H|T]) :- et(H,_), pConsulta14_01(T).
pConsulta14_01([H|T]) :- vfl(H,_), pConsulta14_01(T).
pConsulta14_01([H|T]) :- id(H,_,ID), print('<NoVFLnET id="'),
print(ID), print('"/>'), nl, pConsulta14_01(T).
```

- **Q17** - Return the headwords of the entries which contain a certain word (``hockey").

- **XQuery**

```
for $a in input()/dictionary/e
where contains ($a, "hockey")
return
    $a/hwg/hw
```

- **Prolog**

```
pConsulta17 :- findall(Y,e(X,Y),LISTA), pConsulta17_01(LISTA).

pConsulta17_01([]).
pConsulta17_01([H|T]) :- pConsulta17_02(H), pConsulta17_01(T).

pConsulta17_02(ID) :- pConsulta17_hw(ID), !.
pConsulta17_02(ID) :- pConsulta17_pr(ID), !.
pConsulta17_02(ID) :- pConsulta17_pos(ID), !.
pConsulta17_02(ID) :- pConsulta17_vd(ID), !.
pConsulta17_02(ID) :- pConsulta17_vf(ID), !.
pConsulta17_02(ID) :- pConsulta17_cr(ID), !.
pConsulta17_02(ID) :- pConsulta17_def(ID), !.
pConsulta17_02(ID) :- pConsulta17_cr2(ID), !.
pConsulta17_02(ID) :- pConsulta17_def_el(ID), !.
pConsulta17_02(ID) :- pConsulta17_qd(ID), !.
pConsulta17_02(ID) :- pConsulta17_a(ID), !.
pConsulta17_02(ID) :- pConsulta17_w(ID), !.
pConsulta17_02(ID) :- pConsulta17_bib(ID), !.
pConsulta17_02(ID) :- pConsulta17_loc(ID), !.
pConsulta17_02(ID) :- pConsulta17_qt(ID), !.
pConsulta17_02(ID) :- pConsulta17_cr3(ID), !.
pConsulta17_02(ID) :- pConsulta17_i(ID), !.
pConsulta17_02(ID) :- pConsulta17_b(ID), !.
pConsulta17_02(ID) :- pConsulta17_qt_el(ID), !.

pConsulta17_hw(ID) :- findall(X/Y,(hwg(ID,_2),hw(_2,X,Y)),LISTA),
pConsulta17_hw_01(LISTA).
pConsulta17_hw_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), hw(D,ID,B), pHW_C17(D), !.
pConsulta17_hw_01([H|T]) :-  pConsulta17_hw_01(T).
```

```
pConsulta17_pr(ID) :- findall(X/Y,(hwg(ID,_2),pr(_2,X,Y)),LISTA),
pConsulta17_pr_01(LISTA).
pConsulta17_pr_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), pr(D,ID,B), pHW_C17(D), !.
pConsulta17_pr_01([H|T]) :-  pConsulta17_pr_01(T).

pConsulta17_pos(ID) :- findall(X/Y,(hwg(ID,_2),pos(_2,X,Y)),LISTA),
pConsulta17_pos_01(LISTA).

pConsulta17_pos_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), pos(D,ID,B), pHW_C17(D), !.
pConsulta17_pos_01([H|T]) :-  pConsulta17_pos_01(T).

pConsulta17_vd(ID) :- findall(X/Y,(vfl(ID,_2),vd(_2,X,Y)),LISTA),
pConsulta17_vd_01(LISTA).
pConsulta17_vd_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), vd(D,ID,B), vfl(E,D), hwg(E,F),
pHW_C17(F), !.
pConsulta17_vd_01([H|T]) :-  pConsulta17_vd_01(T).

pConsulta17_vf(ID) :- findall(X/Y,(vfl(ID,_2),vf(_2,X,Y)),LISTA),
pConsulta17_vf_01(LISTA).
pConsulta17_vf_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), vf(D,ID,B), vfl(E,D), hwg(E,F),
pHW_C17(F), !.
pConsulta17_vf_01([H|T]) :-  pConsulta17_vf_01(T).

pConsulta17_cr(ID) :- findall(X/Y,(et(ID,_2),cr(_2,X,Y)),LISTA),
pConsulta17_cr_01(LISTA).
pConsulta17_cr_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), cr(D,ID,B), et(E,D), hwg(E,F),
pHW_C17(F), !.
pConsulta17_cr_01([H|T]) :-  pConsulta17_cr_01(T).

pConsulta17_def(ID) :-
```

```
findall(X/Y,(ss(ID,_2),s(_2,_3),def(_3,X,Y)),LISTA),
pConsulta17_def_01(LISTA).
pConsulta17_def_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), def(D,ID,B), s(E,D), ss(F,E), hwg(F,G),
pHW_C17(G), !.
pConsulta17_def_01([H|T]) :-  pConsulta17_def_01(T).

pConsulta17_def_el(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),def(_3,_4),xml/mixedElement(_4,X,Y)),
LISTA), pConsulta17_def_el_01(LISTA).
pConsulta17_def_el_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), xml/mixedElement(D,ID,B), def(E,D),
s(F,E), ss(G,F), hwg(G,I), pHW_C17(I), !.
pConsulta17_def_el_01([H|T]) :-  pConsulta17_def_el_01(T).

pConsulta17_cr2(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),def(_3,_4),cr(_4,X,Y)),LISTA),
pConsulta17_cr2_01(LISTA).
pConsulta17_cr2_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), cr(D,ID,B), def(E,D), s(F,E), ss(G,F),
hwg(G,I), pHW_C17(I), !.
pConsulta17_cr2_01([H|T]) :-  pConsulta17_cr2_01(T).

pConsulta17_qd(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),qd(_5,X,Y)),LISTA)
, pConsulta17_qd_01(LISTA).
pConsulta17_qd_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), qd(D,ID,B), q(E,D), qp(F,E), s(G,F),
ss(I,G), hwg(I,J), pHW_C17(J), !.
pConsulta17_qd_01([H|T]) :-  pConsulta17_qd_01(T).

pConsulta17_a(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),a(_5,X,Y)),LISTA),
pConsulta17_a_01(LISTA).
pConsulta17_a_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), a(D,ID,B), q(E,D), qp(F,E), s(G,F),
```

```
ss(I,G), hwg(I,J), pHW_C17(J), !.
pConsulta17_a_01([H|T]) :-  pConsulta17_a_01(T).

pConsulta17_w(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),w(_5,X,Y)),LISTA),
pConsulta17_w_01(LISTA).
pConsulta17_w_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), w(D,ID,B), q(E,D), qp(F,E), s(G,F),
ss(I,G), hwg(I,J), pHW_C17(J), !.
pConsulta17_w_01([H|T]) :-  pConsulta17_w_01(T).

pConsulta17_bib(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),bib(_5,X,Y)),LISTA
), pConsulta17_bib_01(LISTA).
pConsulta17_bib_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), bib(D,ID,B), q(E,D), qp(F,E), s(G,F),
ss(I,G), hwg(I,J), pHW_C17(J), !.
pConsulta17_bib_01([H|T]) :-  pConsulta17_bib_01(T).

pConsulta17_loc(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),loc(_5,X,Y)),LISTA
), pConsulta17_loc_01(LISTA).
pConsulta17_loc_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), loc(D,ID,B), q(E,D), qp(F,E), s(G,F),
ss(I,G), hwg(I,J), pHW_C17(J), !.
pConsulta17_loc_01([H|T]) :-  pConsulta17_loc_01(T).

pConsulta17_qt(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),qt(_5,X,Y)),LISTA)
, pConsulta17_qt_01(LISTA).
pConsulta17_qt_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), qt(D,ID,B), q(E,D), qp(F,E), s(G,F),
ss(I,G), hwg(I,J), pHW_C17(J), !.
pConsulta17_qt_01([H|T]) :-  pConsulta17_qt_01(T).

pConsulta17_cr3(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),qt(_5,_6),cr(_6,X,
```

```
Y)),LISTA), pConsulta17_cr3_01(LISTA).
pConsulta17_cr3_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), cr(D,ID,B), qt(E,D), q(F,E), qp(G,F),
s(I,G), ss(J,I), hwg(J,L), pHW_C17(L), !.
pConsulta17_cr3_01([H|T]) :-  pConsulta17_cr3_01(T).

pConsulta17_i(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),qt(_5,_6),i(_6,X,Y
)),LISTA), pConsulta17_i_01(LISTA).
pConsulta17_i_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), i(D,ID,B), qt(E,D), q(F,E), qp(G,F),
s(I,G), ss(J,I), hwg(J,L), pHW_C17(L), !.
pConsulta17_i_01([H|T]) :-  pConsulta17_i_01(T).

pConsulta17_b(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),qt(_5,_6),b(_6,X,Y
)),LISTA), pConsulta17_b_01(LISTA).
pConsulta17_b_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), b(D,ID,B), qt(E,D), q(F,E), qp(G,F),
s(I,G), ss(J,I), hwg(J,L), pHW_C17(L), !.
pConsulta17_b_01([H|T]) :-  pConsulta17_b_01(T).

pConsulta17_qt_el(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),qt(_5,_6),xml/mixe
dElement(_6,X,Y)),LISTA), pConsulta17_qt_el_01(LISTA).
pConsulta17_qt_el_01([]).
pConsulta17_qt_el_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), xml/mixedElement(D,ID,B), qt(E,D),
q(F,E), qp(G,F), s(I,G), ss(J,I), hwg(J,L), pHW_C17(L), !.
pConsulta17_qt_el_01([H|T]) :-  pConsulta17_qt_el_01(T).

pHW_C17(ID) :- findall(X,hw(ID,_,X),LISTA), pHW_C17_01(LISTA).
pHW_C17_01([]).
pHW_C17_01([H|T]) :- pLISTA([H],hw), pHW_C17_01(T).
```

- **Q18** - List the headwords of entries which contain a given phrase (``the hockey").

- **XQuery**

```
for $a in input()/dictionary/e
where contains($a, "the hockey")
return
    $a/hwg/hw
```

- **Prolog**

```
pConsulta18 :- findall(Y,e(X,Y),LISTA), pConsulta18_01(LISTA).

pConsulta18_01([]).
pConsulta18_01([H|T]) :- pConsulta18_02(H), pConsulta18_01(T).

pConsulta18_02(ID) :- pConsulta18_hw(ID), !.
pConsulta18_02(ID) :- pConsulta18_pr(ID), !.
pConsulta18_02(ID) :- pConsulta18_pos(ID), !.
pConsulta18_02(ID) :- pConsulta18_vd(ID), !.
pConsulta18_02(ID) :- pConsulta18_vf(ID), !.
pConsulta18_02(ID) :- pConsulta18_cr(ID), !.
pConsulta18_02(ID) :- pConsulta18_def(ID), !.
pConsulta18_02(ID) :- pConsulta18_cr2(ID), !.
pConsulta18_02(ID) :- pConsulta18_def_el(ID), !.
pConsulta18_02(ID) :- pConsulta18_qd(ID), !.
pConsulta18_02(ID) :- pConsulta18_a(ID), !.
pConsulta18_02(ID) :- pConsulta18_w(ID), !.
pConsulta18_02(ID) :- pConsulta18_bib(ID), !.
pConsulta18_02(ID) :- pConsulta18_loc(ID), !.
pConsulta18_02(ID) :- pConsulta18_qt(ID), !.
pConsulta18_02(ID) :- pConsulta18_cr3(ID), !.
pConsulta18_02(ID) :- pConsulta18_i(ID), !.
pConsulta18_02(ID) :- pConsulta18_b(ID), !.
pConsulta18_02(ID) :- pConsulta18_qt_el(ID), !.

pConsulta18_hw(ID) :- findall(X/Y,(hwg(ID,_2),hw(_2,X,Y)),LISTA),
pConsulta18_hw_01(LISTA).
pConsulta18_hw_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), hw(D,ID,B), pHW_C18(D), !.
pConsulta18_hw_01([H|T]) :-  pConsulta18_hw_01(T).
```

```
pConsulta18_pr(ID) :- findall(X/Y,(hwg(ID,_2),pr(_2,X,Y)),LISTA),
pConsulta18_pr_01(LISTA).
pConsulta18_pr_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), pr(D,ID,B), pHW_C18(D), !.
pConsulta18_pr_01([H|T]) :-  pConsulta18_pr_01(T).

pConsulta18_pos(ID) :- findall(X/Y,(hwg(ID,_2),pos(_2,X,Y)),LISTA),
pConsulta18_pos_01(LISTA).
pConsulta18_pos_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), pos(D,ID,B), pHW_C18(D), !.
pConsulta18_pos_01([H|T]) :-  pConsulta18_pos_01(T).

pConsulta18_vd(ID) :- findall(X/Y,(vfl(ID,_2),vd(_2,X,Y)),LISTA),
pConsulta18_vd_01(LISTA).
pConsulta18_vd_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), vd(D,ID,B), vfl(E,D), hwg(E,F),
pHW_C18(F), !.
pConsulta18_vd_01([H|T]) :-  pConsulta18_vd_01(T).

pConsulta18_vf(ID) :- findall(X/Y,(vfl(ID,_2),vf(_2,X,Y)),LISTA),
pConsulta18_vf_01(LISTA).
pConsulta18_vf_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), vf(D,ID,B), vfl(E,D), hwg(E,F),
pHW_C18(F), !.
pConsulta18_vf_01([H|T]) :-  pConsulta18_vf_01(T).

pConsulta18_cr(ID) :- findall(X/Y,(et(ID,_2),cr(_2,X,Y)),LISTA),
pConsulta18_cr_01(LISTA).
pConsulta18_cr_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), cr(D,ID,B), et(E,D), hwg(E,F),
pHW_C18(F), !.
pConsulta18_cr_01([H|T]) :-  pConsulta18_cr_01(T).

pConsulta18_def(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),def(_3,X,Y)),LISTA),
```

```
pConsulta18_def_01(LISTA).
pConsulta18_def_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), def(D,ID,B), s(E,D), ss(F,E), hwg(F,G),
pHW_C18(G), !.
pConsulta18_def_01([H|T]) :-  pConsulta18_def_01(T).

pConsulta18_def_el(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),def(_3,_4),xml/mixedElement(_4,X,Y)),
LISTA), pConsulta18_def_el_01(LISTA).
pConsulta18_def_el_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), xml/mixedElement(D,ID,B), def(E,D),
s(F,E), ss(G,F), hwg(G,I), pHW_C18(I), !.
pConsulta18_def_el_01([H|T]) :-  pConsulta18_def_el_01(T).

pConsulta18_cr2(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),def(_3,_4),cr(_4,X,Y)),LISTA),
pConsulta18_cr2_01(LISTA).
pConsulta18_cr2_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), cr(D,ID,B), def(E,D), s(F,E), ss(G,F),
hwg(G,I), pHW_C18(I), !.
pConsulta18_cr2_01([H|T]) :-  pConsulta18_cr2_01(T).

pConsulta18_qd(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),qd(_5,X,Y)),LISTA)
, pConsulta18_qd_01(LISTA).
pConsulta18_qd_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), qd(D,ID,B), q(E,D), qp(F,E), s(G,F),
ss(I,G), hwg(I,J), pHW_C18(J), !.
pConsulta18_qd_01([H|T]) :-  pConsulta18_qd_01(T).

pConsulta18_a(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),a(_5,X,Y)),LISTA),
pConsulta18_a_01(LISTA).
pConsulta18_a_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), a(D,ID,B), q(E,D), qp(F,E), s(G,F),
ss(I,G), hwg(I,J), pHW_C18(J), !.
```

```
pConsulta18_a_01([H|T]) :-  pConsulta18_a_01(T).

pConsulta18_w(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),w(_5,X,Y)),LISTA),
pConsulta18_w_01(LISTA).
pConsulta18_w_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), w(D,ID,B), q(E,D), qp(F,E), s(G,F),
ss(I,G), hwg(I,J), pHW_C18(J), !.
pConsulta18_w_01([H|T]) :-  pConsulta18_w_01(T).

pConsulta18_bib(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),bib(_5,X,Y)),LISTA
), pConsulta18_bib_01(LISTA).
pConsulta18_bib_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), bib(D,ID,B), q(E,D), qp(F,E), s(G,F),
ss(I,G), hwg(I,J), pHW_C18(J), !.
pConsulta18_bib_01([H|T]) :-  pConsulta18_bib_01(T).

pConsulta18_loc(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),loc(_5,X,Y)),LISTA
), pConsulta18_loc_01(LISTA).
pConsulta18_loc_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), loc(D,ID,B), q(E,D), qp(F,E), s(G,F),
ss(I,G), hwg(I,J), pHW_C18(J), !.
pConsulta18_loc_01([H|T]) :-  pConsulta18_loc_01(T).

pConsulta18_qt(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),qt(_5,X,Y)),LISTA)
, pConsulta18_qt_01(LISTA).
pConsulta18_qt_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), qt(D,ID,B), q(E,D), qp(F,E), s(G,F),
ss(I,G), hwg(I,J), pHW_C18(J), !.
pConsulta18_qt_01([H|T]) :-  pConsulta18_qt_01(T).

pConsulta18_cr3(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),qt(_5,_6),cr(_6,X,
Y)),LISTA), pConsulta18_cr3_01(LISTA).
```

```
pConsulta18_cr3_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), cr(D,ID,B), qt(E,D), q(F,E), qp(G,F),
s(I,G), ss(J,I), hwg(J,L), pHW_C18(L), !.
pConsulta18_cr3_01([H|T]) :-  pConsulta18_cr3_01(T).

pConsulta18_i(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),qt(_5,_6),i(_6,X,Y
)),LISTA), pConsulta18_i_01(LISTA).
pConsulta18_i_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), i(D,ID,B), qt(E,D), q(F,E), qp(G,F),
s(I,G), ss(J,I), hwg(J,L), pHW_C18(L), !.
pConsulta18_i_01([H|T]) :-  pConsulta18_i_01(T).

pConsulta18_b(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),qt(_5,_6),b(_6,X,Y
)),LISTA), pConsulta18_b_01(LISTA).
pConsulta18_b_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), b(D,ID,B), qt(E,D), q(F,E), qp(G,F),
s(I,G), ss(J,I), hwg(J,L), pHW_C18(L), !.
pConsulta18_b_01([H|T]) :-  pConsulta18_b_01(T).

pConsulta18_qt_el(ID) :-
findall(X/Y,(ss(ID,_2),s(_2,_3),qp(_3,_4),q(_4,_5),qt(_5,_6),xml/mixe
dElement(_6,X,Y)),LISTA), pConsulta18_qt_el_01(LISTA).
pConsulta18_qt_el_01([]).
pConsulta18_qt_el_01([H|T]) :- format(atom(Atom), '~w', [H]),
sub_atom(Atom,_,_,_,'the hockey'), atomic_list_concat([A|[B|C]], '/',
Atom), atom_to_term(A,ID,_), xml/mixedElement(D,ID,B), qt(E,D),
q(F,E), qp(G,F), s(I,G), ss(J,I), hwg(J,L), pHW_C18(L), !.
pConsulta18_qt_el_01([H|T]) :-  pConsulta18_qt_el_01(T).

pHW_C18(ID) :- findall(X,hw(ID,_,X),LISTA), pHW_C18_01(LISTA).
pHW_C18_01([]).
pHW_C18_01([H|T]) :- pLISTA([H],hw), pHW_C18_01(T).
```

- **Q19** - Retrieve the headwords of entries cited, in etymology part, by certain entry with id attribute value (E1).

- **XQuery**

```
for $ent in input()/dictionary/e[@id="E1"],
    $related in input()/dictionary/e
where $ent/et/cr = $related/@id
return
    <Output>
        {$related/hwg/hw}
    </Output>
```

- **Prolog**

```
pConsulta19 :- id(X,_,'E1'), et(X,Y),
findall(CR,cr(Y,_,CR),LISTA_CR), pConsulta19_01(LISTA_CR).
pConsulta19_01([]).
pConsulta19_01([H|T]) :-
findall(V,(id(Z,_,H),hwg(Z,U),hw(U,_,V)),LISTA_HW),
print('<Output>'), nl, pConsulta19_02(LISTA_HW), print('</Output>'),
nl, pConsulta19_01(T).
pConsulta19_02([]).
pConsulta19_02([H|T]) :- print('<hw>'), print(H), print('</hw>'), nl,
pConsulta19_02(T).
```

1.3 MULTIPLE DOCUMENT TEXT CENTRIC XQUERY QUERIES

In this section we present the multiple document/text centric XQuery queries and the corresponding Prolog queries.

- **Q1 -** Return the title of the article that has matching id attribute value (1).

- **XQuery**

```
for $art in input()/article[@id="1"]
return
   $art/prolog/title
```

- **Prolog**

```
pConsulta01 :- id(_1,_,'1'), prolog(_1,_2), title(_2,_,T),
print('<title>'), print(T), print('</title>').
```

- **Q2 -** Find the title of the article authored by (Ben Yang).

  - **XQuery**

```
for $prolog in input()/article/prolog
where
   $prolog/authors/author/name="Ben Yang"
return
   $prolog/title
```

  - **Prolog**

```
pConsulta02 :-
setof(_2,(prolog(_1,_2),authors(_2,_3),author(_3,_4),aname(_4,_,'Ben
Yang')),LISTA), pConsulta02_01(LISTA), !.
pConsulta02 :- true.
pConsulta02_01([]).
pConsulta02_01([H|T]) :- title(H,_,TITLE), print('<title>'),
print(TITLE), print('</title>'), nl, pConsulta02_01(T).
```

- **Q3 -** Group articles by date and calculate the total number of articles in each group.

  - **XQuery**

```
for $a in distinct-values (input()/article/prolog/dateline/date)
let $b := input()/article/prolog/dateline[date=$a]
return
    <Output>
        <Date>{$a/text()}</Date>
        <NumberOfArticles>{count($b)}</NumberOfArticles>
    </Output>
```

  - **Prolog**

```
pConsulta03 :-
setof(X,(_1^_2^_3^_4^_5^(article(_1,_2),prolog(_2,_3),dateline(_3,_4)
,date(_4,_5,X))),LISTA), pConsulta03_01(LISTA).
pConsulta03_01([]).
pConsulta03_01([H|T]) :- findall(_2,date(_1,_2,H),LISTA2),
length(LISTA2,LENGTH), print('<Output>'), nl, print('<Date>'),
print(H), print('</Date>'), nl, print('<NumberOfArticles>'),
```

```
print(LENGTH), print('</NumberOfArticles>'), nl, print('</Output>'),
nl, pConsulta03_01(T).
```

- **Q4 -** Find the heading of the section following the section entitled “Introduction” in a certain article with id attribute value (8).

  - **XQuery**

```
for $a in input()/article[@id="8"]/body/section
    [@heading="introduction"],
    $p in input()/article[@id="8"]/body/section
    [. >> $a][1]
return
    <HeadingOfSection>
        {$p/@heading}
    </HeadingOfSection>
```

  - **Prolog**

```
pConsulta04 :- findall(W,(article(X,Y), id(Y,_,'8'), body(Y,Z),
section(Z,W)),LISTA), pConsulta04_01(LISTA).
pConsulta04_01([]).
pConsulta04_01([H|T]) :- heading(H,_,'introduction'),
pConsulta04_02(T), !.
pConsulta04_01([H|T]) :- pConsulta04_01(T).
pConsulta04_02([]).
pConsulta04_02([DESIRED|REST]) :- heading(DESIRED,_,HEADING),
print('<HeadingOfSection>'), nl, print(HEADING), nl,
print('</HeadingOfSection>').
```

- **Q5 -** Return the headings of the first section of a certain article with id attribute value (9).

  - **XQuery**

```
for $a in input()/article[@id="9"]
return
    <HeadingOfSection>
        {$a/body/section[1]/@heading}
    </HeadingOfSection>
```

- **Prolog**

```
pConsulta05 :- id(X,Y,'9'), body(X,Z), section(Z,W),
heading(W,_,HEADING), !, print('<HeadingOfSection heading="'),
print(HEADING), print('"/>') .
```

- **Q6 -** Find titles of articles where both keywords (``the'' and ``hockey'') are mentioned in the same paragraph of abstracts.

- **XQuery**

```
for $a in input()/article
where some $b in $a/body/abstract/p satisfies
    (contains($b, "the") and contains($b, "hockey"))
return
    $a/prolog/title
```

- **Prolog**

```
pConsulta06 :- findall(_2,article(_1,_2),LISTA),
pConsulta06_01(LISTA).
pConsulta06_01([]).
pConsulta06_01([H|T]) :-
findall(_5,(body(H,_3),abstract(_3,_4),p(_4,_,_5)),LISTA_1),
pConsulta06_02(LISTA_1,H), pConsulta06_01(T).
pConsulta06_02([H2|T2],K) :- sub_atom(H2,_,_,_,'the'),
sub_atom(H2,_,_,_,'hockey'), prolog(K,P), title(P,_,TITLE),
print('<title>'), print(TITLE), print('</title>'), nl,!.
pConsulta06_02([H2|T2],K) :- pConsulta06_02(T2,K), !.
pConsulta06_02([],_).
```

- **Q7 -** Find titles of articles where a keyword (``hockey'') is mentioned in every paragraph of abstract.

- **XQuery**

```
for $a in input()/article
where every $b in $a/body/abstract/p satisfies
    contains($b, "hockey")
```

```
return
    $a/prolog/title
```

- **Prolog**

```
pConsulta07 :- findall(_2,article(_1,_2),LISTA),
pConsulta07_01(LISTA).
pConsulta07_01([]).
pConsulta07_01([H|T]) :-
findall(_5,(body(H,_3),abstract(_3,_4),p(_4,_,_5)),LISTA_1),
pConsulta07_02(LISTA_1), prolog(H,PID), title(PID,_,TITLE),
print('<title>'), print(TITLE), print('</title>'), nl,
pConsulta07_01(T), !.

pConsulta07_01([H|T]) :- pConsulta07_01(T).

pConsulta07_02([]).
pConsulta07_02([H2|T2]) :- sub_atom(H2,_,_,_,'hockey'),
pConsulta07_02(T2).
```

- **Q8 -** Return the names of all authors (one element name unknown) of the article with matching id attribute value (2).

- **XQuery**

```
for $art in input()/article[@id="2"]
return
    $art/prolog/*/author/name
```

- **Prolog**

```
pConsulta08 :- id(X,_,'2'), prolog(X,Y), authors(Y,Z),
findall(W,author(Z,W),LISTA), pConsulta08_01(LISTA).
pConsulta08_01([]) :- !.
pConsulta08_01([H|T]) :- aname(H,_,NAME), print('<name>'),
print(NAME), print('</name>'), nl, pConsulta08_01(T).
```

- **Q9 -** Return all author names (several consecutive element unknown) of the article with matching id attribute value (3).

- **XQuery**

```
for $art in input()/article[@id="3"]
return
   $art//author/name
```

- **Prolog**

```
pConsulta09 :- id(X,_,'3'), prolog(X,Y), authors(Y,Z),
findall(W,author(Z,W),LISTA), pConsulta09_01(LISTA).
pConsulta09_01([]) :- !.
pConsulta09_01([H|T]) :- aname(H,_,NAME), print('<name>'),
print(NAME), print('</name>'), nl, pConsulta09_01(T).
```

- **Q10 -** List the titles of articles sorted by country.

- **XQuery**

```
for $a in input()/article/prolog
order by $a/dateline/country
return
    <Output>
        {$a/title}
        {$a/dateline/country}
    </Output>
```

- **Prolog**

```
pConsulta10 :- findall(C,(article(A,B), prolog(B,C)),LISTA_NO_DATE),
findall(E-C,(article(A,B), prolog(B,C), dateline(C,D),
country(D,_,E)), LISTA), keysort(LISTA,ORDERED),
pConsulta10_01(LISTA_NO_DATE), printListAfterKeySort(ORDERED).

pConsulta10_01([]).
pConsulta10_01([H|T]) :- not(dateline(H,DATELINE)), title(H,_,TITLE),
print('<Output>'), nl, print('<title>'), print(TITLE),
print('</title>'), nl, print('</Output>'), nl, pConsulta10_01(T), !.
pConsulta10_01([H|T]) :- pConsulta10_01(T).

printListAfterKeySort([]).
```

```
printListAfterKeySort([X-H|T]) :- title(H,_,TITLE),
print('<Output>'), nl, print('<title>'), print(TITLE),
print('</title>'), nl, print('<country>'), print(X),
print('</country>'), nl,  print('</Output>'), nl,
printListAfterKeySort(T).
```

- **Q11 -** List the titles of articles that have a matching country element type (Canada), sorted by date.

  - **XQuery**

```
for $a in input()/article/prolog
where $a/dateline/country="Canada"
order by $a/dateline/date
return
    <Output>
        {$a/title}
        {$a/dateline/date}
    </Output>
```

  - **Prolog**

```
pConsulta11 :- findall(E-C,(article(A,B), prolog(B,C), dateline(C,D),
country(D,_,'Canada'), date(D,_,E)), LISTA), keysort(LISTA,ORDERED),
printListAfterKeySort(ORDERED).

printListAfterKeySort([]).
printListAfterKeySort([X-H|T]) :- title(H,_,TITLE),
print('<Output>'), nl, print('<title>'), print(TITLE),
print('</title>'), nl, print('<date>'), print(X), print('</date>'),
nl, print('</Output>'), nl, printListAfterKeySort(T).
```

- **Q12 -** Retrieve the body of the article that has a matching id attribute value (4).

  - **XQuery**

```
for $a in input()/article[@id="4"]
return
    <Article>
        {$a/body}
```

```
    </Article>
```

- **Prolog**

```
pConsulta12 :- id(ID,_,'4'), print('<Article>'), nl, pBody(ID),
print('</Article>').

pBody(ID) :- findall(P,(body(ID,B), abstract(B,A), p(A,_,P)),LISTA),
findall(S,(body(ID,B),section(B,S)), LISTA2), print('<body>'), nl,
print('<abstract>'), nl, pParagraphs(LISTA), print('</abstract>'),
nl, pSection(LISTA2), print('</body>').

pParagraphs([]).
pParagraphs([H|T]) :- print('<p>'), print(H), print('</p>'), nl,
pParagraphs(T).

pSection([]).
pSection([H|T]) :-  findall(P,p(H,_,P),LISTA3),
findall(S,subsec(H,S),LISTA4), heading(H,_,HEADING), print('<section
heading="'), print(HEADING), print('">'), nl, pParagraphs(LISTA3),
pSubsec(LISTA4), print('</section>'), nl, pSection(T).

pSubsec([]).
pSubsec([H|T]) :- findall(P,p(H,_,P),LISTA3),
findall(S,subsec(H,S),LISTA4), heading(H,_,HEADING), print('<subsec
heading="'), print(HEADING), print('">'), nl, pParagraphs(LISTA3),
pSubsec(LISTA4), print('</subsec>'), nl, pSubsec(T).
```

- **Q13 -** Construct a brief information on the article that has a matching id attribute value (5), including title, the name of first author, date and abstract.

- **XQuery**

```
for $a in input()/article[@id="5"]
return
    <Output>
        {$a/prolog/title}
        {$a/prolog/authors/author[1]/name}
        {$a/prolog/dateline/date}
        {$a/body/abstract}
    </Output>
```

- **Prolog**

```
pConsulta13 :- id(X,_,'5'), prolog(X,Y), title(Y,_,TITLE),
authors(Y,Z), author(Z,W), aname(W,_,NAME),
dateline(Y,U),date(U,_,DATE), body(X,V), print('<Output>'), nl,
print('<title>'), print(TITLE), print('</title>'), nl,
print('<name>'), print(NAME), print('</name>'), nl, print('<date>'),
print(DATE), print('</date>'), nl, findall(A,abstract(V,A),LISTA),
((not(LISTA=[]), print('<abstract>'), pConsulta13_01(LISTA),
print('</abstract>'), print('</Output>'));print('</Output>')).
      pConsulta13_01([]).
pConsulta13_01([H|T]) :- p(H,_,P), print('<p>'), print(p),
print('</p>'), nl, pConsulta13_01(T).
```

- **Q14 -** List article title that doesn't have genre element.

- **XQuery**

```
for $a in input()/article/prolog
where empty ($a/genre)
return
    <NoGenre>
        {$a/title}
    </NoGenre>
```

- **Prolog**

```
pConsulta14 :- findall(Y,prolog(X,Y),LISTA), pConsulta14_01(LISTA).

pConsulta14_01([]).
pConsulta14_01([H|T])  :-  not(genre(H,_,GENRE)),  title(H,_,TITLE),
print('<NoGenre>'),      nl,      print('<title>'),      print(TITLE),
print('</title>'),  nl,   print('</NoGenre>'),  nl,  pConsulta14_01(T),
!.
pConsulta14_01([H|T]) :- pConsulta14_01(T).
```

- **Q15 -** List author names whose contact elements are empty in articles.

- **XQuery**

```
for $a in input()/article/prolog/authors/author
where empty($a/contact/text())
return
    <NoContact>
        {$a/name}
    </NoContact>
```

- **Prolog**

```
pConsulta15 :- findall(Y,aname(X,_,Y),LISTA), pConsulta15_01(LISTA).

pConsulta15_01([]).
pConsulta15_01([H|T]) :- print('<NoContact>'), nl, print('<name>'),
print(H), print('</name>'), nl, print('</NoContact>'), nl,
pConsulta15_01(T).
```

- **Q16 -** Get the article by its id attribute value (6).

- **XQuery**

```
for $a in input()/article[@id="6"]
return
    $a
```

- **Prolog**

```
pConsulta16 :- pArticle('6').

pArticle(ID) :- id(X,_,ID), lang(X,_,LANG), print('<article id="'),
print(X), print('" lang="'), print(LANG), print('">'), nl,
pProlog(X), pBody(X), pEpilog(X), print('</article>').


pProlog(ID) :- prolog(ID,X), print('<prolog>'), nl, title(X,_,TITLE),
print('<title>'), print(TITLE), print('</title>'), nl,
((authors(X,Y), findall(Z,author(Y,Z),LISTA), print('<authors>'), nl,
pAuthors(LISTA), print('</authors>'), nl);not(authors(X,Y))),
pDateLine(X), ((genre(X,_,GENRE), print('<genre>'), print(GENRE),
print('</genre>'), nl);not(genre(X,_,GENRE))), ((keywords(X,W),
findall(U,keyword(W,_,U),LISTA2), print('<keywords>'), nl,
pKeywords(LISTA2), print('</keywords'), nl);not(keywords(X,W))),
```

```
print('</prolog>').


pAuthors([]).
pAuthors([H|T]) :- aname(H,_,NAME), print('<author>'), nl,
print('<name>'), print(NAME), print('</name>'), nl, pContact(H),
print('</author>'), nl, pAuthors(T).

pContact(ID) :- (contact(ID,X), print('<contact>'), nl,
((email(X,_,Y), print('<email>'), print(Y), print('</email>'), nl,
((phone(X,_,Z), print('<phone>'), print(Z), print('</phone>'),
nl);not(phone(X,_,Z))));not(email(X,_,Y)), ((phone(X,_,Z),
print('<phone>'), print(Z), print('</phone>'),
nl);not(phone(X,_,Z)))), print('</contact>'), nl) ;
not(contact(ID,X)).

pDateLine(ID) :- dateline(ID,X), print('<dateline>'), nl,
city(X,_,Y), country(X,_,Z), date(X,_,W), print('<city>'), print(Y),
print('</city>'), nl, print('<country>'), print(Z),
print('</country>'), nl, print('<date>'), print(W), print('</date>'),
nl, print('</dateline>'), nl.

pKeywords([]).
pKeywords([H|T]) :-  print('<keyword>'), print(H),
print('</keyword'), nl, pKeywords(T).

pBody(ID) :- findall(P,(body(ID,B), abstract(B,A), p(A,_,P)),LISTA),
findall(S,(body(ID,B),section(B,S)), LISTA2), print('<body>'), nl,
print('<abstract>'), nl, pParagraphs(LISTA), print('</abstract>'),
nl, pSection(LISTA2), print('</body>'), nl.


pParagraphs([]).
pParagraphs([H|T]) :- print('<p>'), print(H), print('</p>'), nl,
pParagraphs(T).

pSection([]).
pSection([H|T]) :-  heading(H,_,HEADING), findall(P,p(H,_,P),LISTA3),
findall(S,subsec(H,S),LISTA4), print('<section heading="'),
print(HEADING), print('">'), nl, pParagraphs(LISTA3),
pSubsec(LISTA4), print('</section>'), nl, pSection(T).
```

```
pSubsec([]).
pSubsec([H|T]) :- heading(H,_,HEADING), findall(P,p(H,_,P),LISTA3),
findall(S,subsec(H,S),LISTA4), print('<subsec heading="'),
print(HEADING), print('">'), nl, pParagraphs(LISTA3),
pSubsec(LISTA4), print('</subsec>'), nl, pSubsec(T).


pEpilog(ID) :- epilog(ID,X), print('<epilog>'), nl,
findall(Z,(acknowledgements(X,Y),pa(Y,_,Z)),LISTA1),
findall(V,(references(X,U),a_id(U,_,V)),LISTA2),
print('<acknowledgements>'), nl, pPA(LISTA1),
print('</acknowledgements>'), nl, print('<references>'), nl,
pAID(LISTA2), print('</references>'), nl, print('</epilog>'), nl.

pPA([]).
pPA([H|T]) :- print('<pa>'), print(H), print('</pa>'), nl, pPA(T).

pAID([]).
pAID([H|T]) :- print('<a_id>'), print(H), print('</a_id>'), nl,
pAID(T).
```

- **Q17** - Return the titles of articles which contain a certain word (“hockey”).

  - **XQuery**

```
for $a in input()/article
where contains ($a//p, "hockey")
return
    $a/prolog/title
```

  - **Prolog**

```
pConsulta17 :- findall(Y,article(X,Y),LISTA), pConsulta17_01(LISTA).

pConsulta17_01([]).

pConsulta17_01([H|T]) :- body(H,A),
findall(P1,(abstract(A,_1), p(_1,_,P1)), LISTA1),
findall(P2,(section(A,_2), p(_2,_,P2)),LISTA2),
findall(P3,(section(A,_2), subsec(_2,_3), p(_3,_,P3)),LISTA3),
findall(P4,(section(A,_2), subsec(_2,_3), subsec(_3,_4),
```

```
p(_4,_,P4)),LISTA4),
findall(P5,(section(A,_2), subsec(_2,_3), subsec(_3,_4),
subsec(_4,_5), p(_5,_,P5)),LISTA5),
append(LISTA1,LISTA2,LISTA_TMP1),
append(LISTA_TMP1,LISTA3,LISTA_TMP2),
append(LISTA_TMP2,LISTA4,LISTA_TMP3),
append(LISTA_TMP3,LISTA5,LISTA_FINAL), findP(LISTA_FINAL),
prolog(H,PROLOG_ID), title(PROLOG_ID,_,TITLE), print('<title>'),
print(TITLE), print('</title>'), nl, pConsulta17_01(T), !.

pConsulta17_01([H|T]) :- pConsulta17_01(T).

findP([]) :- false.
findP([H|T]) :- sub_atom(H,_,_,_,'hockey'), !.
findP([H|T]) :- findP(T).
```

- **Q18** - List the titles and abstracts of articles which contain a given phrase (“the hockey”).

  - **XQuery**

```
for $a in input()/article
where contains ($a//p, "the hockey")
return
    <Output>
        {$a/prolog/title}
        {$a/body/abstract}
    </Output>
```

  - **Prolog**

```
pConsulta18 :- findall(Y,article(X,Y),LISTA), pConsulta18_01(LISTA).

pConsulta18_01([]).

pConsulta18_01([H|T]) :- body(H,A),
findall(P1,(abstract(A,_1), p(_1,_,P1)),LISTA1),
findall(P2,(section(A,_2), p(_2,_,P2)),LISTA2),
findall(P3,(section(A,_2), subsec(_2,_3), p(_3,_,P3)),LISTA3),
findall(P4,(section(A,_2), subsec(_2,_3), subsec(_3,_4),
p(_4,_,P4)),LISTA4),
```

```
findall(P5,(section(A,_2), subsec(_2,_3), subsec(_3,_4),
subsec(_4,_5), p(_5,_,P5)),LISTA5),
append(LISTA1,LISTA2,LISTA_TMP1),
append(LISTA_TMP1,LISTA3,LISTA_TMP2),
append(LISTA_TMP2,LISTA4,LISTA_TMP3),
append(LISTA_TMP3,LISTA5,LISTA_FINAL),
findP_C18(LISTA_FINAL), prolog(H,PROLOG_ID),
title(PROLOG_ID,_,TITLE), print('<Output>'), nl, print('<title>'),
print(TITLE), print('</title>'), nl,
((not(LISTA1=[]), print('<abstract>'), nl, pParagraphs(LISTA1),
print('</abstract>'), nl, print('</Output>'), nl,
pConsulta18_01(T));(print('</Output>'), nl, pConsulta18_01(T))), !.

pConsulta18_01([H|T]) :- pConsulta18_01(T).

findP_C18([]) :- false.
findP_C18([H|T]) :- sub_atom(H,_,_,_,'the hockey'), nl, !.
findP_C18([H|T]) :- findP_C18(T).
```

- **Q19 -** List the names of articles cited by an article with a certain id attribute value (7).

  - **XQuery**

```
for $a in input()/article[@id='7']/epilog/references/a_id,
   $b in input()/article
where $a = $b/@id
return
   <Output>
      {$b/prolog/title}
   </Output>
```

  - **Prolog**

```
pConsulta19 :- id(ID,_,'7'), findall(AID,(epilog(ID, EPILOG_ID),
references(EPILOG_ID,REF_ID), a_id(REF_ID,_,AID)),LISTA),
pConsulta19_01(LISTA).

pConsulta19_01([]).
pConsulta19_01([H|T]) :- id(X,_,H), prolog(X,Y), title(Y,_,TITLE),
print('<Output>'), nl, print('<title>'), print(TITLE),
print('</title>'), nl, print('</Output>'), nl, pConsulta19_01(T).
```