PROLOG

DATA EXTRACTION

WITH DOZENS OF EMBEDDED DEDICATED PREDICATES

- Text files container.
- Iterating among text files.
- Precise positioning inside text files.
- Marking area(s) inside text file.
- Cuting marked area.
- Extracting desired text. Various algorithms.
- Storing extracted data. Various ways.
- Building knowledge basis with extracted data.
- Organizing, shaping, reshaping data.
- Collecting, (re)mapping, (pre)packing.
- Integrating data with other sources.
- Building SQLite3 DB with or from that data.
- Using data from SQLite3 and other RDB.

Content

Introduction

An example of collecting information from stored text files using my Prolog and for this purpose specially developed predicates. Files are stored as html pages that contain information. Roughly areas that are recognized as the basis for the development of dedicated predicates:

- Text files container.
- Iterating among text files.
- Precise positioning inside text files.
- Marking area(s) inside text file.
- Cutting marked area.
- Extracting desired text. Various algorithms.
- Storing extracted data. Various ways.
- Building knowledge basis with extracted data.
- Organizing, shaping, and reshaping data.
- Collecting, (re)mapping, (pre)packing.
- Integrating data with other sources.
- Building SQLite3 DB with or from that data.
- Using data from SQLite3 and other RDB.

Every recognized situation got predicate that solves or helps solving its problem. Often in several versions.

Also every further using most probably will result with new predicates.

Data representation is not covered in this document: it is separate area that has its own explanation.

What is in this document are various situations that usually occur in data extraction, collection, organization… and one question that I meet almost always , mostly first one in some new conversation:

First question:

As for a fully automated process, something truly autonomous, what answers could you offer primarily in terms of data extraction, collection, organization, distribution, valorization… but not only in these areas?

The tasks

Case 1:

Build Prolog knowledge base from obtained text files according to desired data shape. For this case let it be:

rdf(ClubName,"ImaID",ClubID)

and

rdf(ClubID,AgeCategory,NameSurname)

Case 2:

The extracted data should be reshaped according to some new condition.

Make data shape that will contain seniors age category from desired club , then list it on screen and save it on disk.

Case 3:

The extracted data should be grouped according to age categories. All members of a category would be saved in a file named by that category.

However, the names of the categories in the extracted space are not suitable for the new functionality, so they need to be changed according to the provided exchanging map.

Also exact number of category members is required.

Case 4:

The extracted data should be aggregated in one file.

Every line must contain Category, Club and Name.

All of lines must be ordered alphabetically by the name of Club.

Case 5:

Design a database with 3 table: Members, Clubs and Category. Make appropriate foreign keys. Table Members must have Name and Surname fields, so existing data must be first atomized then properly organized.

Reshape extracted data to suitable corresponding predicate.

Load database.

Show query execution on the databases thus obtained using various programs.

What are the 3 most common surnames in the data obtained?

How often is my name in this data?

About cutting algorithm

Imagine reading a text. In essence, there are two theoretical possibilities:

- The eyes are in the same place while the paper is moving or,
- The eyes are moving while the paper remains its position.

In one moment the eyes will find and recognize a piece of text that may be needed for something. That future data needs to be taken from that environment and moved to another one designed for some purpose. So, in order to take that data we may use a number of algorithms. All of them will spent some time to explained eyes or paper movement until find and recognize what they are looking for. It would be of great help to somehow reduce that time.

The idea is based on identifying the place where the interesting part begins. If we cut of everything before that, every next search will not have to waste time in that area.

And the recognition itself will be significantly facilitated as there will be far fewer options to check.

Cutting example

An example, a text containing information about clubs:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<title>Hrvatski judo savez  - klubovi</title>
<meta name="keywords" content="sport hjs cro judo union" />
<meta name="description" content="Web stranica Hrvatskog judo saveza" />
<link href="default.css" rel="stylesheet" type="text/css" media="screen" />
<script type="text/javascript" src="scripts/jquery-1.8.3.min.js"></script>
<script type="text/javascript" src="scripts/gallery.js"></script>
<script type="text/javascript" src="slide.js"></script>
<script type="text/javascript" src="jquery.cycle.all.js"></script>
<script type="text/javascript">

  var _gaq = _gaq || [];
  _gaq.push(['_setAccount', 'UA-19028542-1']);
  _gaq.push(['_trackPageview']);

  (function() {
    var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;
    ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-analytics.com/ga.js';
    var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
  })();

  $(document).ready(function() {
    $('.slide_sponzori').cycle({


                        fx: 'fade' // choose your transition type, ex: fade, scrollUp, shuffle, etc...
                });
});
```

```
</script>
</head>
<body>
<!-- glava -->
<div id="logo">
</div>
<div id="menu">
		<ul id="main">
			<li class="current_page_item_static">
			<table cellpadding="0" cellspacing="0" width="100%" class="menu_left_links">
			<tr>
				<td align="center"><a href="http://www.eju.net" target="_blank">
				  <img src="img/hnav/EJU.jpg" border="0" alt="Evropski judo savez" title="Evropski judo savez" /></a>
			    </td>
				<td align="center"><a href="http://www.ijf.org" target="_blank">
					<img src="img/hnav/IJF.jpg" border="0" alt="Svijetski judo savez" title="Svijetski judo savez" /></a>
				</td>
				<td align="center"><a href="http://www.hoo.hr/" target="_blank">
					<img src="img/hnav/HOO.jpg" border="0" alt="Hrvatski olimpijski odbor" title="Hrvatski olimpijski odbor"
/></a>
				</td>
			</tr>
			</table>
			</li>
			<li><a href="./?page=entry">POČETNA</a></li><li><a href="./?page=content">JUDO SAVEZ</a></li><li><a
href="./?page=news">VIJESTI</a></li><li><a href="./?page=com">NATJECANJA</a></li><li class="current_page_item"><a href="./?page=clubs">KLUBOVI</a></li>
			<li class="current_page_item_konec" style="vertical-align:middle;">
			</li>
		</ul>
</div>
<!-- konec glave -->
<!-- zacetek strani -->
<div id="page">
	<!-- levi stolpec -->
	<div id="sidebar1" class="sidebar">
	  <ul>
<li><h2>Suci i treneri</h2>
    <ul>
     <li><a href="./?page=clubs&sub=judge&club=-1"  >Suci</a></li>
     <li><a href="./?page=clubs&sub=coach&club=-1"  >Treneri</a></li>
    </ul>
    </li><li><h2>Klubovi</h2><ul><li><a href="./?page=clubs&sub=detail&club=155" >AJK Mladost
Zagreb</a></li><li><a href="./?page=clubs&sub=detail&club=221" >AJK Student</a></li><li><a
href="./?page=clubs&sub=detail&club=241" >Belišće</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000016" >Biokovo Makarska</a></li><li><a
href="./?page=clubs&sub=detail&club=2107000023" >Black Belt</a></li><li><a
href="./?page=clubs&sub=detail&club=249" >Braća Radić</a></li><li><a
href="./?page=clubs&sub=detail&club=2119000049" >Bushido</a></li><li><a
href="./?page=clubs&sub=detail&club=197" >Centar</a></li><li><a href="./?page=clubs&sub=detail&club=212"
>Dalmacijacement</a></li><li><a href="./?page=clubs&sub=detail&club=198" >Dojo Zagreb</a></li><li><a
href="./?page=clubs&sub=detail&club=246" >Dubrovnik</a></li><li><a href="./?page=clubs&sub=detail&club=247"
>Dubrovnik 1966</a></li><li><a href="./?page=clubs&sub=detail&club=245" >Duga Resa</a></li><li><a
href="./?page=clubs&sub=detail&club=248" >Dugave</a></li><li><a
href="./?page=clubs&sub=detail&club=2107000021" >HJS</a></li><li><a
href="./?page=clubs&sub=detail&club=163" >Ippon Labin</a></li><li><a
href="./?page=clubs&sub=detail&club=162" >Istarski borac Pula</a></li><li><a
href="./?page=clubs&sub=detail&club=481" >Istra</a></li><li><a href="./?page=clubs&sub=detail&club=191"
>Jaska Jastrebarsko</a></li><li><a href="./?page=clubs&sub=detail&club=2103000006" >JSDNZ</a></li><li><a
href="./?page=clubs&sub=detail&club=2103000007" >JSIZ</a></li><li><a
href="./?page=clubs&sub=detail&club=2112000035" >JSPGZ</a></li><li><a
href="./?page=clubs&sub=detail&club=2103000005" >JSSDZ</a></li><li><a
href="./?page=clubs&sub=detail&club=2112000034" >JSZZ</a></li><li><a
href="./?page=clubs&sub=detail&club=240" >Judokan Požega</a></li><li><a
href="./?page=clubs&sub=detail&club=239" >Karlovac</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000105" >Kaštela</a></li><li><a
href="./?page=clubs&sub=detail&club=510" >Kokica</a></li><li><a href="./?page=clubs&sub=detail&club=238"
>Konavle Cavtat</a></li><li><a href="./?page=clubs&sub=detail&club=237" >Krk</a></li><li><a
href="./?page=clubs&sub=detail&club=236" >Kutjevo</a></li><li><a
href="./?page=clubs&sub=detail&club=2119000047" >Kvarner</a></li><li><a
href="./?page=clubs&sub=detail&club=180" >Lika Zagreb</a></li><li><a
href="./?page=clubs&sub=detail&club=190" >Maksimir Zagreb</a></li><li><a
href="./?page=clubs&sub=detail&club=2118000004" >Medvedgrad</a></li><li><a
href="./?page=clubs&sub=detail&club=2140000011" >Meto Labin</a></li><li><a
href="./?page=clubs&sub=detail&club=235" >Mihovil</a></li><li><a href="./?page=clubs&sub=detail&club=177"
>Mladost Osijek</a></li><li><a href="./?page=clubs&sub=detail&club=233" >Mokošica</a></li><li><a
href="./?page=clubs&sub=detail&club=232" >Nijemci</a></li><li><a href="./?page=clubs&sub=detail&club=486"
>Nippon</a></li><li><a href="./?page=clubs&sub=detail&club=231" >Novi Zagreb</a></li><li><a
href="./?page=clubs&sub=detail&club=479" >Okruk</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000001" >Omiš</a></li><li><a
href="./?page=clubs&sub=detail&club=2112000025" >Osijek</a></li><li><a
```

```
href="./?page=clubs&sub=detail&club=156" >Panda</a></li><li><a href="./?page=clubs&sub=detail&club=229"
>Panda Pušća</a></li><li><a href="./?page=clubs&sub=detail&club=178" >Pinky</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000132" >Pisarovina</a></li><li><a
href="./?page=clubs&sub=detail&club=228" >Policijska akademija</a></li><li><a
href="./?page=clubs&sub=detail&club=482" >Profectus Samobor</a></li><li><a
href="./?page=clubs&sub=detail&club=154" >Pujanke</a></li><li><a href="./?page=clubs&sub=detail&club=196"
>Pulafit</a></li><li><a href="./?page=clubs&sub=detail&club=225" >Randori Pula</a></li><li><a
href="./?page=clubs&sub=detail&club=2107000019" >Remete</a></li><li><a
href="./?page=clubs&sub=detail&club=184" >Rijeka</a></li><li><a href="./?page=clubs&sub=detail&club=447"
>Sakura</a></li><li><a href="./?page=clubs&sub=detail&club=192" >Samobor</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000151" >Samurai</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000018" >Sesvete</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000005" >Šibenik</a></li><li><a
href="./?page=clubs&sub=detail&club=224" >Slano</a></li><li><a href="./?page=clubs&sub=detail&club=223"
>Sokol</a></li><li><a href="./?page=clubs&sub=detail&club=222" >Solin</a></li><li><a
href="./?page=clubs&sub=detail&club=213" >Split</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000213" >Sveučilište u Dubrovniku</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000189" >Sveučilište u Osijeku</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000205" >Sveučilište u Rijeci</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000186" >Sveučilište u Splitu</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000185" >Sveučilište u Zagrebu</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000021" >Tempo</a></li><li><a
href="./?page=clubs&sub=detail&club=226" >Uchimata</a></li><li><a href="./?page=clubs&sub=detail&club=220"
>Ura nage</a></li><li><a href="./?page=clubs&sub=detail&club=2110000130" >Val</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000188" >Veleučilište u Vukovaru</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000187" >Veleučilište u Zagrebu</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000190" >Veleučilište Velika Gorica</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000212" >Veleučilište VERN</a></li><li><a
href="./?page=clubs&sub=detail&club=227" >Veteran</a></li><li><a href="./?page=clubs&sub=detail&club=250"
>Vinkovci</a></li><li><a href="./?page=clubs&sub=detail&club=2110000207" >Visoka Škola za primjenjeno
računarstvo</a></li><li><a href="./?page=clubs&sub=detail&club=219" >Zadar</a></li><li><a
href="./?page=clubs&sub=detail&club=182" >Zagreb</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000129" >Zagrebačka Judo Škola</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000206" >Zdravstveno Veleučilište Zagreb</a></li><li><a
href="./?page=clubs&sub=detail&club=2103000004" >ZJS</a></li><li><a
href="./?page=clubs&sub=detail&club=480" >Zrinski Čakovec</a></li><li><a
href="./?page=clubs&sub=detail&club=2110000026" >Župa Dubrovačka</a></li></ul></li></ul>
      </div>
      <!-- konec levega stolpca -->

            <!-- vsebine -->

                                    <div id="widecontent">
                  <script>
            $(document).ready(function() {
                  $('#st_tek').html( $('#st-tek_skupno').text() );
            });

</script>

<div class="post">
      <h1 class="title"><a href="./?page=clubs">Klubovi Hrvatskog Judo Saveza</a></h1>
      <br>
      <table cellpadding=2>
        <tr><td>klubovi</td><td>88</td></tr>
        <tr><td>natjecatelja</td><td><span id="st_tek"></span></td></tr>
        <tr><td>suci</td><td><a href="./?page=clubs&sub=judge&club=-1">33</a></td></tr>
        <tr><td>treneri</td><td><a href="./?page=clubs&sub=coach&club=-1">119</a></td></tr>
      </table>
    </div><div class="post"><table class="tabHead"><tr><th>uzrast</th><th>broj natjecatelja</th></tr><tr><td><a
href="./?page=clubs&sub=stat&kat=1">seniori</a></td><td align=center>914</td></tr><tr><td><a href="./?page=clubs&sub=stat&kat=2">seniorke</a></td><td
align=center>225</td></tr><tr><td><a href="./?page=clubs&sub=stat&kat=23">juniori U21</a></td><td align=center>531</td></tr><tr><td><a
href="./?page=clubs&sub=stat&kat=24">juniorke U21</a></td><td align=center>185</td></tr><tr><td><a href="./?page=clubs&sub=stat&kat=25">kadeti U18</a></td><td
align=center>679</td></tr><tr><td><a href="./?page=clubs&sub=stat&kat=26">kadetkinje U18</a></td><td align=center>228</td></tr><tr><td><a
href="./?page=clubs&sub=stat&kat=27">ml. kadeti U16</a></td><td align=center>992</td></tr><tr><td><a href="./?page=clubs&sub=stat&kat=28">ml. kadetkinje
U16</a></td><td align=center>354</td></tr><tr><td><a href="./?page=clubs&sub=stat&kat=29">dječaci U14</a></td><td align=center>1373</td></tr><tr><td><a
href="./?page=clubs&sub=stat&kat=30">curice U14</a></td><td align=center>444</td></tr><tr><td><a href="./?page=clubs&sub=stat&kat=31">dječaci U12</a></td><td
align=center>1615</td></tr><tr><td><a href="./?page=clubs&sub=stat&kat=32">curice U12</a></td><td align=center>527</td></tr><tr><td><a
href="./?page=clubs&sub=stat&kat=19">limači U10</a></td><td align=center>1539</td></tr><tr><td><a href="./?page=clubs&sub=stat&kat=20">curice U10</a></td><td
align=center>449</td></tr><tr><td><a href="./?page=clubs&sub=stat&kat=21">limači U8</a></td><td align=center>707</td></tr><tr><td><a
href="./?page=clubs&sub=stat&kat=22">curice U8</a></td><td align=center>151</td></tr>      <tr bgcolor="#B2B2B2" style="border-bottom:1px solid #CCCCCC"><td>Ukupno
muški</td><td align="center">8350</td></tr>
                  <tr bgcolor="#B2B2B2" style="border-bottom:1px solid #CCCCCC"><td>Ukupno žene</td><td align="center">2563</td></tr>
                  <tr bgcolor="#B2B2B2" style="border-bottom:1px solid #CCCCCC"><td>Ukupno</td><td align="center"><span id="st-tek_skupno">10913</span></td></tr>
                </table></div></div>
                <div style="clear: both;"> </div>
</div>
<!-- konec strani -->
<div id="footer">
<p>Svi materijali na ovoj stranici zaštićeni su autorskim pravom. Svako kopiranje i neovlašteno preuzmanje sadržaja biti će utuženo po zakonu o autorskim pravima.</p>
            <p>&copy; 2009 by <a href="mailto: info@datastat.si">DataStat d.o.o.</a></p>
</div>
</body>
</html>
```

Code for cutting example

Only the red labeled part of text contains interesting information. Let's get started with cutting some of the text we do not need. So let's first load the text file:

```
Prolog Interpreter v12.0 Nov 23 2015
DVP textin('example1.txt').
12410 characters loaded.
OK
[1 solutions; 12.528 cpu; heap 0 bytes]
```

Then we make predicate that indicate which part of the text we are interested in:

```
DVP cct(X):- after('</li><li><h2>Klubovi</h2><ul><li><ahref="./?',X,1)
,before('<!-- konec levega stolpca -->',X,1).
DVP cct(0).
OK
[1 solutions; 1 inferences; 0.002 cpu; 0.5 klips; heap 22 bytes]
*
```

So all behind „
<h2>Klubovi</h2>
<ahref="./?'“ and in front „<!-- konec levega stolpca -->“ . The rest is cut of.

Now we have moved our paper at right place. Let's see how our saved text starts now:

**page=clubs&sub=detail&club=155" >AJK Mladost Zagreb
AJK Student
Belišće
**

The program goes as follows:

```
nl :- write('\n').
DVP ardf(X,Y,Z):-extract('page=clubs&sub=detail&club=','" >',0,Z),
   extract('" >','</a></li><li>',0,X),
   Y='ImaID',
    write(rdf(X,Y,Z)),nl,
    rdfin(X,Y,Z),
    ardf(X1,Y1,Z1).
DVP ardf(QX1,QY1,QZ1).
```

Extract to me the information that is located between:

„page=clubs&sub=detail&club=“

and

„ " > „

When extracting, remove the left marker, so that the text after the extraction starts from the right marker. In this case, information about club ID was obtained at that way.

Next, extract the information that is between:

„" >“

and

„
„

This is now the name of the club. It should be noted that a marker

„" >“

without cutting would be recognized so many times if it would have been started from the beginning of the original text. As soon as we get closer to the information and cut the ballast, we are more confident that we can use a smaller number of characters for the marker itself.

Marker does not need to be part of text at all. Can be added to text; when viewing information on a web site, or at some other way, You can choose what separates the necessary information from unnecessary without entering the text itself and searching for suitable markers. Then cutting is not necessary, but it certainly accelerates parsing.

The upper cutting is explained here. It can also be lower and could be done without cutting: after/3 is marked as the beginning of the search and moved with extracting information instead of cutting, until the before/3 mark. This way is more demanding for resources and for users who must find suitable markers in the text that do not appear elsewhere if it is for other reasons important to start from the beginning oh the file. Otherwise search and cut could start from the end of file, using the same reasoning as described.

Predicate for extraction…

…used in following Case 1 example.

after/3 after(Marker, DocumentNumber, cutYesNo)

Succeed if finds a marker regardless of whether cut is ordered or not. By the logic of things, it cuts all up to marker including the marker itself.

before/3 before(Marker, DocumentNumber, cutYesNo)

Succeed if finds a marker regardless of whether cut is ordered or not. By the logic of things, it cuts all after marker including the marker itself.

extract/4 extract(MarkerBefore,MarkerAfter, DocumentNumber, HoldExtractedValue)

Succeed if finds both markers and cuts everything up to the second marker.

textin/1 textin('example1.txt'). textin('./judoke').

Succeed if finds a file and saves it in the input files field. If the entry is directory, succeed if manages to load all the files in the directory. Then report about every particular name, how many characters it contains and how much they all have characters.

textInInfo/1 textInInfo(X)

Succeed if there are loaded files. Sets a variable to the last index of uploaded files enabling iteration on document fields at that way.

rdfin/3 `rdfin(Subject,Predicat,Object)`

Succeed if put on heap a new fact. So, in principle, it always works if there is enough memory. What really rdfin/3 actually do? Dynamically add a new fact, as if this program item was executed:

```
rdf(Subject,Predicat,Object):-true.
```

This fact is on disposal immediately.

fin/n `fin(Functor,Term1,Term2,…)`

A variant that allows any functor name, not just rdf, and any number of arguments. Also dynamically add a new fact:

```
functor(Term1,Term2,…):-true.
```

There are dozens more dedicated predicates from the extract family. One group is based on the exact number of characters that can be identified by a marker in a copy or some other way.

The second has a list as expected result and returns anything between the specified characters, for example between and throughout the document or some marked area, etc...

Real power is combination of some of them that are suitable for some concrete situation

Case 1 - extracting data

The case starts with a situation in which, for one competition, quick access to data on judokas from the national judo federation had to be made.

Data source were organized at following way: first examined page holds just clubs names. Others, 88 of them, contain information, among others, about name, surname, age category and gender but not club name. So, some connection between club name and data on page which describes that club must be established. ID from URL is the best candidate.

Once obtained the data must be prepared for various purpose. A few hypothetical cases will be exposed after the most important one - the extraction of the data.

The task:

Build Prolog knowledge base from obtained text files according to desired data shape.

For this case let it be:

```
rdf(ClubName,"ImaID",ClubID)
```

and

```
rdf(ClubID,AgeCategory,NameSurname)
```

The code:

The code listed below, along with the comments, needs to be copy /pasted in Prolog.

It will: load 88 files and find the starting information at each location: club name and id. Then it starts from the beginning, parses other documents where using already stored information and add it to the appropriate triple as new facts. After all, we have rdf repository with 10500+ judokas, their clubs, ...

```
textin('example1.txt').
textin('./judoke').
nl :- write('\n').
ardf(X,Y,Z):-extract('page=clubs&sub=detail&club=','" >',0,Z),
                  extract('" >','</a></li>',0,X),
                   Y='ImaID',
                   write(rdf(X,Y,Z)),nl,
         rdfin(X,Y,Z),
       ardf(X1,Y1,Z1).


c(X):- after('Natjecatelji',X,1),
before('<!-- konec strani -->',X,1),
textInInfo(TP), S is X+1, S<TP, c(S).

c(1).
```

/*

With "after" and "before" we define what from each page should be cut, and it will be repeated through all the uploaded files. ***Only red labeled parts from above example will remain.*** *The TP shows how many loaded files are, so c (x) repeats until the number is reached.*

*/

```
natjecatelji(ID,Num):-
after('page=clubs&sub=comp&club=', Num,1),
extract('">','</a></td>',Num,Ime),
extract('<td>','</td>',Num,Kat),
 write(rdf(ID,Kat,Ime)),nl,
rdfin(ID,Kat,Ime), !,natjecatelji(ID,Num).
natjecatelji(ID,Num):- true.
```

/*

The iteration procedure for the loaded page. Takes the name of the club and the index number of the loaded page, extracts the requested information (name of the judo and the age category), prints the triplet, stores it in the memory and ... that there is no such thing as this "cut" is, that is, that the program is finished, each page loads only one data because there would be no iterations per page. If you cut a cut, at one point the program would stop executing because one of the extracts would not succeed. So, until everything is done properly up to the "cut", iteration must continue, as soon as it cannot be repeated, that something from the extracts fails, the goal is fulfilled.

*/

```
a(X):- write(X ), nl, nl,
extract('page=clubs&sub=comp&club=','&cat=1" >',X,Z),
```

```
write(X - Z), nl,
rdf(Klub,_,Z),
write(Klub - Z), nl,
after('<th>prezime i ime</th><th>uzrastna kat</th>',X,1),
natjecatelji(Klub,X),
textInInfo(TP), S is X+1, write(S), nl, S<TP, a(S).
```

/*

Iteration procedure for all loaded pages. First, a page index with two new rows is printed to make it visible on the printout, then find the ID of the club that page contain. The page index and the club ID (to visualize the described process) are printed, so rdf prompts the name of the club by its ID. Then it is printed. Cut a part of the text that is the excess, which does not have the name of the judokas, and the procedure is called (the contestants) natjecatelji / 2 with the received club name and the page index being processed. Then iterates through all the pages.

*/

```
ardf(QX1,QY1,QZ1).
b(Xh):- textInInfo(TnP), write(TnP), nl, a(Xh).
b(1).
/*
```

Execute a procedure that loads the names of clubs in triplets. Defines a wrapper around a(X) that serves to further track iteration across all pages. It is also called upon to execute.

```
*/
```

We now have a Prolog knowledge base filled with extracted data.

This is the starting point for all the cases described below.

Case 2 – simple reshape and save data

The task:

The extracted data should be reshaped according to some new condition.

Make data shape that will contain seniors age category from desired club, then list it on screen and save it on disk.

```
rdf(Klub,'curice U12',_).
rdf(Klub,'seniori',_), rdf(Klub,_,'155').

newshape(Name,Klub):-rdf(Klub,'seniori',Name), rdf(Klub,_,'155').
newshape(Name,Klub).
```

Some examples of queries performed on the generated Prolog knowledge base.

Take a look at QueryListing to see how they look like.

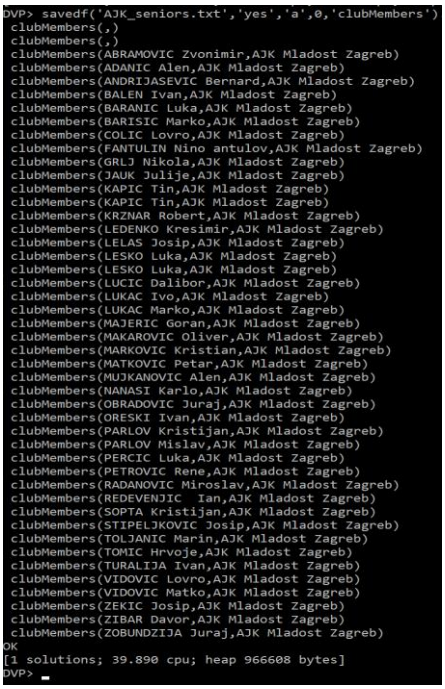
Instead of just listing some reshaped data, we could both list and save:

```
rdf(Klub,'seniori',Name), rdf(Klub,_,'155'),fin(clubMembers,Name,Klub).
savedf('AJK_seniors.txt','yes','a',0,'clubMembers').
```

`savedf(path,saveNosave,mod,num,functor)/5`

Succeed if there is no deformation of the data in the specified part of the knowledge base and if it stores the collected data successfully. There are 15 ways to use it.

fin(clubMembers,Name,Klub) means fin(functorName,X1,X2, ...)/n is used to dynamically make knowledge base. In fact it does the same thing as described rdf but has possibility to define functor name and take multiple arguments.



savedf(path,saveNosave,mod,num,functor)/5 is powerful predicate that allows both console listing and saving.

Path defines file to save, saveNosave with num determines saving or listing and how many entries will be used or avoided.

Functor could be a list of functors that contains entries which are about to be saved and/or listed.

So, what we have on left picture is listing of reshaped data, just seniors from club along with club name. In the same time it is also saved in the very same form in AJK_seniors.txt file. Generated files could be zipped and shared on various ways.

Case 3 – data grouping and multiple streams

It will rarely happen in practice that such a simple reshaping is the desired state of the data. Grouping data by some criteria is a very common situation.

Second common situation is remapping data from current definition to another, demanded one.

Because re-mapping could be quite complicated, and so often occurs, a special approach to this situation greatly facilitates actual programming.

One of embedded types in my Prolog is **bind-type**. It is, in essence, map generalization. Its representation on heap is by three consecutive position in which first two are some other types and third one is description of its behaviour. That behaviour is separated in three cases: unifying, printing (on screen or in file...) and anything other what most commonly means logic and arithmetic function.

By generating a bind-type it has “instructions” on how to behave in which situation. There are 35 possible situations by which is determined how to use which member: which one will be printed, which one unified and so on.

The task:

The extracted data should be grouped according to age categories. All members of a category would be saved in a file named by that category.

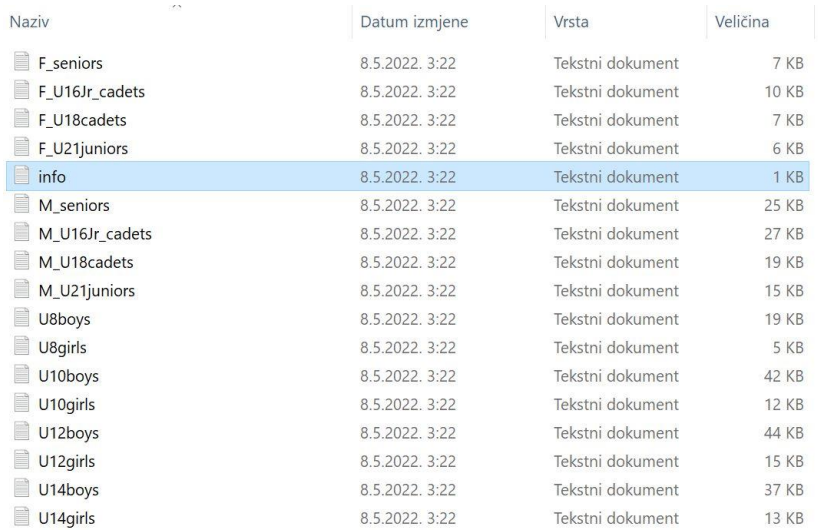
However, the names of the categories in the extracted space are not suitable for the new functionality, so they need to be changed according to the provided exchanging map.

Also exact number of category members is required.

The code:

```
counter('M_seniors'(0), 'F_seniors'(0), 'M_U21juniors'(0),
'F_U21juniors'(0), 'M_U18cadets'(0), 'F_U18cadets'(0),
'M_U16Jr_cadets'(0), 'F_U16Jr_cadets'(0), 'U14boys'(0), 'U14girls'(0),
'U12boys'(0), 'U12girls'(0), 'U10boys'(0),
'U10girls'(0),'U8boys'(0), 'U8girls'(0),members(0)).


/* EXTRACTING CODE DESCRIBED IN CASE 1    */


getCat(V):-rdf(_,X,_),X\=='ImaID',list(addU,X,V).
getCategory(V):-getCat(V),!,getCategory(V).
getCategory(V):-!.


getData(KlubName,Kat):- rdf(Klub,Kat,Name),
rdf(Klub,'ImaID',_),Kat\=='ImaID', concat2(Klub,' - ',Name,KlubName).


set(openWrite, "arrangementByCategories/info.txt", info, a, print).


next12(X):- concat("arrangementByCategories/",X,".txt",X1),
set(openWrite, X1, X, a, print).


next25(X,XX,Kat,KlubName):- X==Kat, Kl=X,
writeTerm(Kl,KlubName),inc(Kl),inc(members),
set(forEachBreak,25,XX).


Cat=[M_seniors, F_seniors, M_U21juniors, F_U21juniors,
M_U18cadets, F_U18cadets, M_U16Jr_cadets, F_U16Jr_cadets,
U14boys, U14girls, U12boys, U12girls, U10boys, U10girls,
  U8boys, U8girls],foreach12(Cat),next12(X),printSolutionsFalse,
set(noBacktraceArea,1000000),getCategory(V),
set(bTL,V,Cat,L3),set(defaultBind,17,M),
getData(KlubName,Kat),foreach25(L3),next25(X,XX,Kat,KlubName),
reset(forEachBreak,25).


printSolutionsTrue,use(addVariablePrint),inc(all,KLl),
writeTerm(info,'*******************************  info
*******************************\n'),
writeTerm(info,KLl),
writeTerm(info,'*******************************  info
*******************************\n'),
set(closeWrite, print).
```

This is the content of ordered arrangementByCategories folder after program execution:



Naziv	Datum izmjene	Vrsta	Veličina
F_seniors	8.5.2022. 3:22	Tekstni dokument	7 KB
F_U16Jr_cadets	8.5.2022. 3:22	Tekstni dokument	10 KB
F_U18cadets	8.5.2022. 3:22	Tekstni dokument	7 KB
F_U21juniors	8.5.2022. 3:22	Tekstni dokument	6 KB
info	8.5.2022. 3:22	Tekstni dokument	1 KB
M_seniors	8.5.2022. 3:22	Tekstni dokument	25 KB
M_U16Jr_cadets	8.5.2022. 3:22	Tekstni dokument	27 KB
M_U18cadets	8.5.2022. 3:22	Tekstni dokument	19 KB
M_U21juniors	8.5.2022. 3:22	Tekstni dokument	15 KB
U8boys	8.5.2022. 3:22	Tekstni dokument	19 KB
U8girls	8.5.2022. 3:22	Tekstni dokument	5 KB
U10boys	8.5.2022. 3:22	Tekstni dokument	42 KB
U10girls	8.5.2022. 3:22	Tekstni dokument	12 KB
U12boys	8.5.2022. 3:22	Tekstni dokument	44 KB
U12girls	8.5.2022. 3:22	Tekstni dokument	15 KB
U14boys	8.5.2022. 3:22	Tekstni dokument	37 KB
U14girls	8.5.2022. 3:22	Tekstni dokument	13 KB

Note marked info.txt

It contains required information about number of members of each category. Every file contains list of all category member. Look, for example, the content of F_seniors.txt.

Such a short code, but with 17 built-in predicates characteristic for my Prolog. In fact, this small sample of code contains a very small part of the usual look of Prolog that a Prolog developer would immediately recognize.

So, before start explaining the code, we will first look at the basic description of the predicate.

Some embedded Predicates

counter/n `counter(cnt1(0), cnt2(10), …)`

Succeed if there is enough memory to set the counter, which means always because the definition of the counter must be first among all other predicates. After the name of the counter is its initialization, the starting number from which the counting will begin.

inc/2 inc(members) , inc("F-seniors",FS), inc(All,A)

Succeed allways because what it does is just incrementing existing value of named counter. If has two arguments then shows current value of targeted counter or, if first one is "All", shows values of all registered counters.

list(addU,X,V)/3 list(addUnique, memberToAdd, list)

Succeed if memberToAdd does not exists in composed list. Then new member will be added to list. Limited functionality due to backtracing.

concat/n concat(part1, part2, ..., Concatenated)

Succeed if parts are strings itself, means could be evaluated as character arrays. Last term in predicate is variable that will contain concatenated string from all parts. Theoretical number of parts could be up to 256. Resulting string will be stored as atom name.

concat2/n concat2(part1, part2, ..., Concatenated)

Same as concat except resulting string will be stored in char container.

set(openWrite, path, info, a, print)/5

Succeed if named stream will be successfully opened. This is writing stream, it will use provided path, "a" means append, "print" means show us yours message (opening, closing, error...), and "info" is his name. All predicates that works with outstreams expect just its name.

next25(X,XX,Kat,KlubName)/n

Succeed allways if logic conditions in his body are satisfied. next(00-99) is predicate what is called from corresponding foreach(00-99) that is traversing through anything iterable. First argument is provided by foreach, second, if present, is number of iterations up to that moment, and any other is transferred from body of current predicate that contain foreach-next pair.

foreach25(X)/1-8

Succeed if internal conditions for setting up the iteration is successful. Could have 1-8 arguments depending of situation and iterable entity. In one run up to 99 different foreach-next pairs can be nested in any possible way.

set(forEachBreak,25,XX)/3

Succeed allways. In this case it is just signal for foreach25 then its next25 has reached conditions and there is no need to iterate any more.

reset(forEachBreak,25).

Succeed allways. In this case it is just signal for foreach25 to, in case of new iteration, do not stop at previous break point.

writeTerm(Kl,KlubName)/2-4

Succeed if named stream and provided term are writable. Could have 2 to 4 arguments depending of situation and used term.

set(noBacktraceArea,1000000)/2

Succeed if is possible to set up new heap area. In essence this predicate allows some area of heap to be permanent in use, to accumulate without backtracing and this fact create new possibilities. For example mentioned list(addU,X,V)/3 could now be used across all predicates at any number of runs and collect its list at same place. Extremely useful and really fast.

set(bTL,V,Cat,L3)/4-5 bTL means bindTypeList

Succeed if binding is possible. This predicate construct list L3 as bind type list by binding every member of list V to corresponding member of list Cat. Since it has 4 arguments it uses defultBinding. Fifth argument defines binding that is not default.

set(defaultBind,17,M)/3

Succeed always. In this case it changes global binding to situation 17 (use first, unify first, print second) and in variable M shows previous state.

printSolutionsTrue / False

Succeed allways. This is a directive for the Prolog engine to print or not print solutions on the screen. Non-prints are much faster, and if they will be written to disk anyway, you don't have to watch them on the screen either.

use(addVariablePrint)/1

Succeed always. Directive for adding external variables (in this case counters) to printing list.

set(closeWrite, print)/1-2

Succeed if the streams were successfully closed. Must be executed in order of flushing all opened streams.

Code explanation

Global counters must be defined first and they retain their values throughout the program.

Calling inc (something) will increase that something by default or set value. For example, in following code snippet:

```
next25(X,XX,Kat,KlubName):- X==Kat, Kl=X,
writeTerm(Kl,KlubName),inc(Kl),inc(members), . . .
```

inc(Kl) will increment that particular counter that is provided from foreach25 to its next25 (variable X that is assigned to variable Kl, and this variable is category name) and also counter members. So, counter `members` will be incremented in any call while other counters just when its matching condition `X==Kat` will be true.

inc(something,ShowMe) will not increment counter something but rather shows its current value.

```
DVP> counter('M_seniors'(0), 'F_seniors'(0), 'M_U21juniors'(0), 'F_U21juniors'(0), 'M_U18cadets'(0), 'F_U18cadets'(0),
'M_U16Jr_cadets'(0), 'F_U16Jr_cadets'(0), 'U14boys'(0), 'U14girls'(0), 'U12boys'(0), 'U12girls'(0), 'U10boys'(0),
'U10girls'(0),'U8boys'(0), 'U8girls'(0),members(0)).
Counter M_seniors = 0 is set
Counter F_seniors = 0 is set
Counter M_U21juniors = 0 is set
Counter F_U21juniors = 0 is set
Counter M_U18cadets = 0 is set
Counter F_U18cadets = 0 is set
Counter M_U16Jr_cadets = 0 is set
Counter F_U16Jr_cadets = 0 is set
Counter U14boys = 0 is set
Counter U14girls = 0 is set
Counter U12boys = 0 is set
Counter U12girls = 0 is set
Counter U10boys = 0 is set
Counter U10girls = 0 is set
Counter U8boys = 0 is set
Counter U8girls = 0 is set
Counter members = 0 is set
OK
[1 solutions; 14.275 cpu; heap 0 bytes]
DVP>
```

Counters will show itself by creation. They could be incremented, decremented, seted, reseted and showed.

Their state could be used by program at any time of execution. There is also the possibility to show them all::

```
VP> inc(all,KLl).
Ll = [10526,144,654,425,1488,518,1575,440,1347,348,968,220,664,183,523,198,831]
_seniors = 831
_seniors = 198
_U21juniors = 523
_U21juniors = 183
_U18cadets = 664
_U18cadets = 220
_U16Jr_cadets = 968
_U16Jr_cadets = 348
14boys = 1347
14girls = 440
12boys = 1575
12girls = 518
10boys = 1488
10girls = 425
8boys = 654
8girls = 144
embers = 10526
```

This is the way info.txt were created.

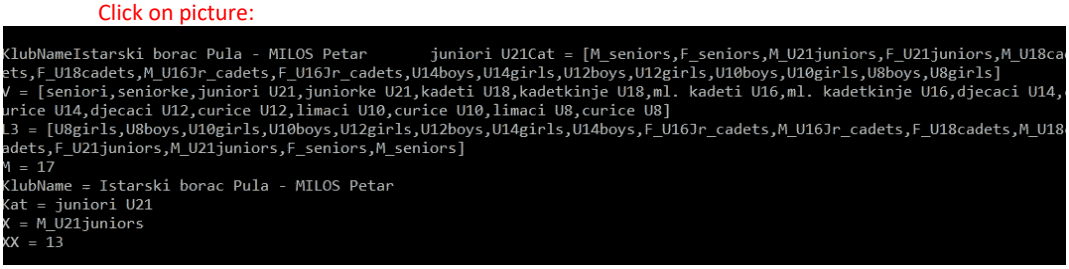
The content of mentioned info.txt file:

```
WRITE Stream:"info" opened at: 2022-05-08 03:20:29:014

******************************  info ********************************


[10526,144,654,425,1488,518,1575,440,1347,348,968,220,664,183,523,198,831]

M_seniors = 831

F_seniors = 198

M_U21juniors = 523

F_U21juniors = 183

M_U18cadets = 664

F_U18cadets = 220

M_U16Jr_cadets = 968

F_U16Jr_cadets = 348

U14boys = 1347

U14girls = 440

U12boys = 1575

U12girls = 518

U10boys = 1488

U10girls = 425

U8boys = 654

U8girls = 144

Members = 10526


******************************  info ********************************


WRITE Stream:" info " closed at:  2022-05-08 03:22:19:761  Elapsed time:
00:01:50.747
```

The WRITE Stream called “info” describes itself with the exact moments of engagement. Its content, list of counters state, is requested information of category members number.

Let's dwell for a while on the aforementioned code snippet. This is a standard nesting loop with extracted data that could be visualized:

Click on picture:



That exchange also shows a kind of polymorphism. In the moment of comparison:

```
next25(X,XX,Kat,KlubName):- X==Kat, Kl=X,

writeTerm(Kl,KlubName),inc(Kl),inc(members),

set(forEachBreak,25,XX).
```

Its bind type nature allows, let say, appropriate behaviour to current situation.

Bind type X, according to its state 17, will compare its first member with Kat, and if match assign itself to variable Kl (binding, unless forced, does not work with variables). But, state 17 says that for printing is second member used, so writeTerm will get the same binded X and use second member.

Consider now binding state 27 instead of 17 (force use first, force unify second, print second). In this case variable Kl will get value of second X member and writeTerm will not have nothing to choose because is getting simple plain string, not bind type.

In both cases mapping is performed by bind type behaviour.

Case 4 – data ordering and aggregating

The task:

The extracted data should be aggregated in one file.
Every line must contain Category, Club and Name.
All of lines must be ordered alphabetically by the name of Club.

Slightly modified previous task will use SQLite3 database as ordering engine, even it is easy resolvable without that help.

The code:

```
getCat(V):-rdf(_,X,_),X\=='ImaID',list(addU,X,V).
getCategory(V):-getCat(V),!,getCategory(V).
getCategory(V):-!.


getData(Klub,Name,Kat):-
rdf(Klub,Kat,Name), rdf(Klub,'ImaID',_),Kat\=='ImaID'.


set(openWrite, "arrangementByCategories/all2.txt", allByDB, a, print).
set(openSqlite3, ':memory:', iden1,    print).

sql3(iden1,exec,print, 'CREATE TABLE T1(Category CHAR(70),Club
CHAR(70),Name CHAR(70));   ', 1).


next25(X,XX,Kat,Klub,Name):- X==Kat,Kl=X,
concat2("INSERT INTO T1(Category ,Club ,Name)
VALUES('",Kl,"','",Klub,"','",Name,"');",HH),
sql3(iden1,exec,print, HH, 1),set(forEachBreak,25,XX).


Cat=[M_seniors, F_seniors, M_U21juniors, F_U21juniors,
M_U18cadets, F_U18cadets, M_U16Jr_cadets, F_U16Jr_cadets,
U14boys, U14girls, U12boys, U12girls, U10boys, U10girls,
  U8boys, U8girls],printSolutionsFalse,
set(noBacktraceArea,1000000),getCategory(V),
set(bTL,V,Cat,L3),set(defaultBind,17,M),
reset(foreach),getData(Klub,Name,Kat),foreach25(L3),next25(X,XX,Kat,Klub,Na
me),reset(forEachBreak,25).


sql3(iden1, exec, allByDB, "Select * from T1  order by Club asc ", 5, '
\t').

set(closeWrite, print).

set(closeSqlite3, print).
```

My Prolog is fully integrated with SQLite3. Some APIs are available through ad hoc predicates.

Some of SQLite3 embedded predicates

```
set(openSqlite3, ':memory:', iden1,    print).
```

Succeed if creation and/or connection with SQLite3 DB was successful. In this case DB is created in memory, identification of this particular connection is “iden1” and all possible messages will be printed on screen.

```
sql3(iden1,exec,print,’querry’ , callback).
```

Succeed if standard SQLite3 exec function succeeds with executing query. On possible error messages will be forwarded to screen. It could be dosens of connections opened and queries through all of them could be executed in parallel.

```
sql3(iden1, exec, allByDB, ’querry’, 5, ' \t').
```

Succeed if standard SQLite3 exec function succeeds with executing query. For this overloaded version is characteristic that third argument is valid write stream, so query result will be forwarded to it. Last argument is delimiter among records.

```
set(closeSqlite3, print).
```

Succeed if DB is successfully closed.

Code explanation

Using the same code base as previous example Categories, Clubs and Name are obtained. The foreachXX/nextXX paradigm is explained in the previous example. The second type of loop, which looks more like a Prolog species, will be described in detail here.

```
getCat(V):-rdf(_,X,_),X\=='ImaID',list(addU,X,V).
getCategory(V):-getCat(V),!,getCategory(V).
getCategory(V):-!.
```

getCat is traversing rdf by getting its middle member. If it is not prescribed placeholder 'ImaID' and not already member of constructing list V, will be added to list V. getCategory waits for that moment and call getCat again. When happen that nothing new could enter to constructing list, getCat will not pass getCategory's cut (!) and getCategory will terminate further looping.

Described scenario will not work in others Prologs, at least not in expected way it works here. It must first be blessed with `set(noBacktraceArea,…),` and then an accumulation could occur in the lower heap area. By parser-time-binding, which means that starting heap addresses remains from parsing moment, all goals have their beginnings in the same address area. By execution of `noBacktraceArea` directive they do not lose these entry points and continue to use them regardless of the current address area.

Instead of writing collected data to file, they are loaded in table of in memory created SQLite3 data base.

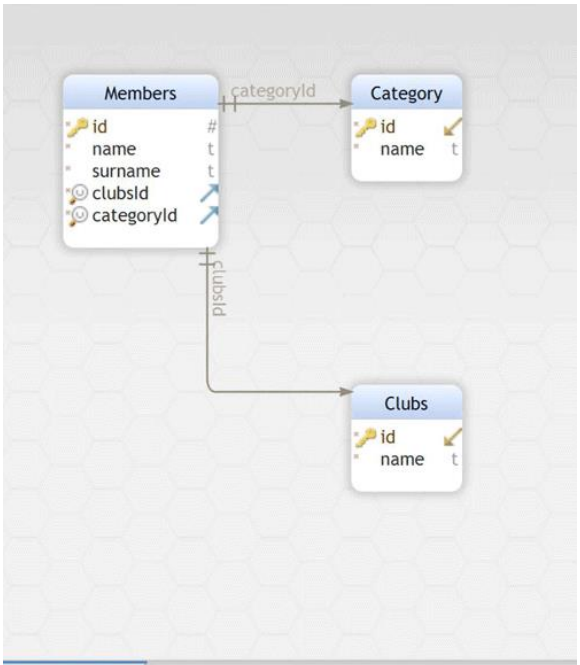
Once the process of table loading finishes, simple select query with ordering by Clubs will be executed.

But, used predicate is "overloaded" so, besides standard SQLite3 exec execution, it uses provided write out stream and delimiter to pack data obtained by SQL query and write it to disk.

So, in one go all the data will be ordered, packed and saved to file.

The result is in file all2.txt .

Case 5 – SQLite3 DB and corresponding predicates



judokeDB.sql

File Edit SQL Run Query Run Script Explain

```
PRAGMA encoding = 'UTF-8';

CREATE TABLE Clubs(
id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
name text NOT NULL
);

CREATE TABLE Category(
id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
name text NOT NULL
);


CREATE TABLE Members (
id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL ,
name text NOT NULL,
surname text NOT NULL,
clubsId INTEGER UNSIGNED NOT NULL,
categoryId INTEGER UNSIGNED NOT NULL,
FOREIGN KEY(clubsId)
    REFERENCES Clubs(id)
      ON DELETE CASCADE
      ON UPDATE NO ACTION,
  FOREIGN KEY(categoryId)
    REFERENCES Category(id)
      ON DELETE CASCADE
      ON UPDATE NO ACTION
);
```

Obtaining data in two differently shaped rdf structure was for exercise purpose only. That approach is not suitable in most cases.

A database must be designed before data collection, but even then using the Prolog structure as an indirect step may be a good idea.

There are several reasons for that. In addition to being reusable with other Prolog capabilities for some tasks, the speed of loading database tables directly from the structure is significantly higher.

This is also truth in opposite direction either: it is much better (easier, faster and more elegant, especially in bigger projects) to load Prolog structure from database in one go.

That is why I have developed special predicates that recognize the internal structure of the Prolog structure and according to it either create a table in the existing database, or, if possible, adjust the data to those expected in the table.

On the left is database for extracted data showed in DbSchema program.

The task:

Design a database with 3 table: Members, Clubs and Category. Make appropriate foreign keys. Table Members must have Name and Surname fields, so existing data must be first atomized then properly organized.

Reshape extracted data to suitable corresponding predicate.

Load database.

Show query execution on the databases thus obtained using various programs.

What are the 3 most common surnames in the data obtained?

How often is my name in this data?

The code:

```
getCat(V):-rdf(_,X,_),X\=='ImaID',list(addU,X,V).
getCategory(V):-getCat(V),!,getCategory(V).
getCategory(V):-!.
```

```
getClub(V):-rdf(Klub,_,_), list(addU,Klub,V).
getClubs(V):-getClub(V),!,getClubs(V).
getClubs(V):-!.
```

```
parseName(A, A1, A2):-list(explode,A," ", 5, Out, Num),
(Num==2,get(nth,Out,0,A1),get(nth,Out,1,A2);
 Num==3,get(nth,Out,0,A11),get(nth,Out,1,A12),
      concat2(A11," ",A12,A1), get(nth,Out,2,A2);
 Num>=4,get(nth,Out,0,A11),get(nth,Out,1,A12),
concat2(A11," ",A12,A1),get(nth,Out,2,A21),
get(nth,Out,3,A22),concat2(A21," ",A22,A2)).
```

```
filterName(IDClub, IDCat, Name):-
  filter(numChars,Name,"-",2, junkRepo,IDClub, IDCat),
   filter(prohibitedChars,Name,"_*",junkRepo,IDClub, IDCat),
    filter(trim,Name).
```

```
next28(X,XX):-fin(clubs,XX,X).
set(noBacktraceArea,5000000),getClubs(V),foreach28(V),next28(X,XX).
```

```
Cat2=["M_seniors", "F_seniors", "M_U21juniors", "F_U21juniors",
"M_U18cadets", "F_U18cadets", "M_U16Jr_cadets", "F_U16Jr_cadets",
"U14boys", "U14girls", "U12boys", "U12girls", "U10boys", "U10girls",
"U8boys", "U8girls"], set(noBacktraceArea,5000000),
getCategory(Vc),lowBindingTrue,set(bTL,Vc,Cat2,27,L3),lowBindingFalse.


next23(X1,XX1):- write(X1),nl, fin(categories,XX1,X1).
foreach23(Cat2),next23(X1,XX1).



 rdf(Xr,A,Y),A\=='ImaID',clubs(Id1,Xr),list(mB,L3,A,AA),categories(Id2,AA),
    parseName(Y, A1, A2),
     filterName(Id1, Id2, A1),
      filterName(Id1, Id2, A2),
        fin(judoke,A2,A1,Id1,Id2).


set(openSqlite3, 'judoke.db3', iden1,    print).

sql3(iden1,fromFile,"judokeDB2.sql").

sql3(iden1,tableAddFromStructure,"Category",categories, 8).
sql3(iden1,tableAddFromStructure,"Clubs",clubs, 8).
sql3(iden1,tableAddFromStructure,"Members",judoke, 261).


sql3(iden1,exec,print, "SELECT Members.id, Members.name, surname, Clubs.name
AS club, Category.name AS category FROM Clubs, Category, Members WHERE
Clubs.id=Members.clubsId AND Category.id=Members.categoryId",1).


set(closeSqlite3, print).
```

```
DVP> sql3(iden1,exec,print, "SELECT * from Category",1).
id                              name
1                               F_seniors
2                               M_U21juniors
3                               F_U21juniors
4                               M_U18cadets
5                               F_U18cadets
6                               M_U16Jr_cadets
7                               F_U16Jr_cadets
8                               U14boys
9                               U14girls
10                              U12boys
11                              U12girls
12                              U10boys
13                              U10girls
14                              U8boys
15                              U8girls
Query executed successfully
NO
[0 solutions; 72.666 cpu; heap 3893770 bytes; 6 vars | 6 rvars ]
```

Running queries with high response on the console does not offer an actual overview of the data. Instead of the query mentioned in the code, this shorter one is executed. The query from the code on the very same database is executed in the HeidiSQL program:

```
SELECT Members.id, Members.name, surname,
Clubs.name AS club,
Category.name AS category
FROM Clubs, Category, Members
WHERE Clubs.id=Members.clubsId
AND Category.id=Members.categoryId
```

Result #1 (10.523r × 5c)

id	name	surname	club	category
1	Josip	BARAC	Pujanke	M_seniors
2	Nikola	DOBRIJEVIC	Pujanke	M_seniors
3	Josip	PISAC	Pujanke	M_seniors
4	Slobodan	PRERADOVIC	Pujanke	M_seniors
5	Ivan	STRIBIC	Pujanke	M_seniors
6	Renato	SUNJERGA	Pujanke	M_seniors
7	Vlatka	KLJAKOVIC GASPIC	Pujanke	F_seniors
8	Dijana	LAUS	Pujanke	F_seniors
9	Marijana	MASTILOVIC	Pujanke	F_seniors
10	Zoran	BLAZEVIC	Pujanke	M_U21juniors
11	Mislav	DURDOV	Pujanke	M_U21juniors
12	Mario	GABRILO	Pujanke	M_U21juniors
13	Ivo	KOVACEVIC	Pujanke	M_U21juniors
14	Andelko	MARETIC	Pujanke	M_U21juniors
15	Jakov	MARETIC	Pujanke	M_U21juniors
16	Tomislav	MIJAILOVIC	Pujanke	M_U21juniors
17	Martin	POJIC	Pujanke	M_U21juniors
18	Jozo	SARIC	Pujanke	M_U21juniors
19	Leon	SURIJA	Pujanke	M_U21juniors
20	Petar	VUKUSIC	Pujanke	M_U21juniors

■ ■ ■

■ ■ ■

■ ■ ■

```
SELECT Members.id, Members.name, surname,
Clubs.name AS club,
Category.name AS category
FROM Clubs, Category, Members
WHERE Clubs.id=Members.clubsId
AND Category.id=Members.categoryId
```

Result #1 (10.523r × 5c)

id	name	surname	club	category
10.504	LOVRO	HECIMOVIC	Kokica	U12boys
10.505	Dzenis	TASAKOVIC	Kokica	U12boys
10.506	Marko	DURASIC	Kokica	U12boys
10.507	Ema	DROBNJAK	Kokica	U12girls
10.508	Lana	JANKOVIC	Kokica	U12girls
10.509	Marta	LUCIC	Kokica	U12girls
10.510	Rita	LUCIC	Kokica	U12girls
10.511	Franka	TOMASIC	Kokica	U12girls
10.512	DAVID	BALEN	Kokica	U10boys
10.513	PATRIK	COSOVIC	Kokica	U10boys
10.514	Andrija	FABIJANCIC	Kokica	U10boys
10.515	ANDREJ	KNEZEVIC	Kokica	U10boys
10.516	Kristijan	KOMPES	Kokica	U10boys
10.517	LUKA	PUTANEC	Kokica	U10boys
10.518	Luka	DURASIC	Kokica	U10boys
10.519	Simona	JANKOVIC	Kokica	U10girls
10.520	Monika	KOMPES	Kokica	U10girls
10.521	Borna	KOMPES	Kokica	U8boys
10.522	FRAN	STOJIC	Kokica	U8boys
10.523	PATRIK	STOJIC	Kokica	U8boys

So, generated SQLite3 database is valid since is wide usable in specialized programs.

More SQLite3 and other embedded predicates

list(explode,A," ", 5, Out, Num)

Succeed if creation of list from provided character array was successful. In this case 5 is maximum length of creating list, so if there will be more records truncate them. Out is created list and Num real number of members of created list.

list(mB,L3,A,AA).

Succeed if find matching. mB means matchBind. Traversing through L3, list of bind type, with comparing all of its members with A. If matching is possible, with regard of current binding, AA and matched L3 member are unified, again with regard to current binding.

get(nth,IterableIn,Num,A).

Succeed if find nth member of iterable entity IterableIn.Num is position of desired element and A is that element, if exists. „Older brother“, get(nnth,…) return n of nth members, list of them…

lowBindingTrue/False

It allways succeeds, but has an impact only if heap address area has been lifted. Directive that set rule of binding: regardless of current address space use lower heap area only. Or stop to do that (True/False).

filter(trim,Name).

One of the filter family predicate. It always succeeds, but has an impact only if there was something to trim. It is not condition filter so always must be used after them.

```
filter(numChars,Name,"-",2, junkRepo,IDClub, IDCat).
```

One of the filter family predicate. Succeed if condition is not satisfied. In this case in observed string Name character "-" could occur less then constrained number (2) of times. If it would happen more times, in junkRepo structure will be saved that particular case along with any number of terms stated after it.

```
filter(prohibitedChars,Name,"_*", junkRepo,IDClub, IDCat).
```

One of the predicates of the filter family. It succeeds if the condition is not met. In this case, the characters "_ *" could not appear at all in the observed sequence. Should this happen, that particular case will be stored in the junkRepo structure along with any number of terms listed after it.

```
sql3(iden1,tableAddFromStructure,"Members",judoke, 261).
```

Succeed if table will successfully be loaded from structure. In this case table Members will be loaded from data in structure judoke. 261 means: use transactions (1), create and add autoincrement id column (4) and use fast option (256), 261=1+4+256. The existing first column can be used as an ID (option 8 used in the code). A complete table can also be created if it does not exist in the database according to the internal relationships of the structure with the specified field names or using the default ones instead. And in this case either it is also possible to choose whether to use the existing first column as an ID or create an autoincrement. Very powerful predicate.

```
sql3(iden1,fromFile,"judokeDB2.sql").
```

Succeed if code was loaded from file and successfully executed. In this case this is only code in file so there is no need for additional terms that would determine internal delimiter and position of desired code snippet.

Code explanation

The above predicate is a good example of "overloaded" predicates. Almost all of them have more than one version that can be intrinsically different, not just syntactically.

Let look at this version:

sql3(iden1,fromFile,"judokeQueries.txt", 2,3).

If judokeQueries.txt file has following content:

```
SELECT Members.id, Members.name, surname,
        Clubs.name AS club, Category.name AS category
        FROM Clubs, Category, Members WHERE
        Clubs.id=Members.clubsId AND Category.id=Members.categoryId//**//
SELECT  Members.name as name, surname,
        Clubs.name AS club, Category.name AS category
        FROM Clubs, Category, Members WHERE
        Clubs.id=Members.clubsId AND Category.id=Members.categoryId
        and Members.name='Damir'//**//
SELECT  surname ,count(*) as number
        FROM Members
        group by surname
        order by count(*) asc//**//
```

This would be the result:

```
VP> sql3(iden1,fromFile,"judokeQueries.txt", 2,1).
ame                     surname                 club                    category
amir                    ZILIC                   Ippon Labin             M_U16Jr_cadets
amir                    IZAKOVIC                Mladost Osijek          U14boys
amir                    KADIC                   Pinky                   M_seniors
amir                    KRSIC                   Samobor                 M_seniors
amir                    KRSIC                   Samobor                 M_seniors
amir                    BESIC                   Pulafit                 M_U18cadets
amir                    BAJAMIC                 Kastela                 M_seniors
amir                    HUSKIC                  Zagrebacka Judo Skola   U12boys
amir                    MILATOVIC               Pisarovina              M_seniors
amir                    CERIMAGIC               Sveuciliste u Zagrebu   M_seniors
amir                    KORUNIC                 Medvedgrad              M_seniors
amir                    OMRCEN                  Dalmacijacement         M_U21juniors
amir                    SEGVIC                  AJK Student             U10boys
amir                    PILJIC                  Nijemci                 M_seniors
amir                    SPOLJARIC               Nijemci                 U14boys
amir                    CERIMAGIC               Dugave                  M_seniors
amir                    STANKOVIC               Vinkovci                M_U21juniors
uery executed successfully
K
1 solutions; 61.205 cpu; heap 0 bytes]
VP>
```

So, if that predicate has more then 3 arguments then 4th is number of desired query among all queries in file and last one is callback function number that determines a way of layout.

But, if 4th term is not number but string, then it determines delimiter and last is number of query assuming default callback. Six arguments means that callback will be also determined.

```
1  SELECT Members.name, surname, Clubs.name AS club,
2  Category.name AS category
3  FROM Clubs, Category, Members
4  WHERE Clubs.id=Members.clubsId
5  AND Category.id=Members.categoryId
6  AND Members.name =  'Damir'
```

Result #1 (18r × 4c)

name	surname	club	category
Damir	OMCIKUS	Pujanke	U8boys
Damir	ZILIC	Ippon Labin	M_U16Jr_cadets
Damir	IZAKOVIC	Mladost Osijek	U14boys
Damir	KADIC	Pinky	M_seniors
Damir	KRSIC	Samobor	M_seniors
Damir	KRSIC	Samobor	M_seniors
Damir	BESIC	Pulafit	M_U18cadets
Damir	BAJAMIC	Kastela	M_seniors
Damir	HUSKIC	Zagrebacka Judo Skola	U12boys
Damir	MILATOVIC	Pisarovina	M_seniors
Damir	CERIMAGIC	Sveuciliste u Zagrebu	M_seniors
Damir	KORUNIC	Medvedgrad	M_seniors
Damir	OMRCEN	Dalmacijacement	M_U21juniors
Damir	SEGVIC	AJK Student	U10boys
Damir	PILJIC	Nijemci	M_seniors
Damir	SPOLJARIC	Nijemci	U14boys
Damir	CERIMAGIC	Dugave	M_seniors
Damir	STANKOVIC	Vinkovci	M_U21juniors

The very same query on the very same database using HeidiSQL.

Even for this small result set it is much more readable then console.

However, there are 18 members with name Damir out of a total of 10,523. This is 0,171%. My name is really rare in that population.

The next question is are all of Damirs here? What if someone's name is Damir-Marko?

That lead us to filtering politics. Jean-Pierre or Villas-Boas are real possibility, so must be taken in consideration.

```
parseName(A, A1, A2):-list(explode,A," ", 5, Out, Num),
Num==2,get(nth,Out,0,A1),get(nth,Out,1,A2);
Num==3,get(nth,Out,0,A11),get(nth,Out,1,A12),
      concat2(A11," ",A12,A1), get(nth,Out,2,A2);
Num>=4,get(nth,Out,0,A11),get(nth,Out,1,A12),
concat2(A11," ",A12,A1),get(nth,Out,2,A21),
get(nth,Out,3,A22),concat2(A21," ",A22,A2)).
```

```
filterName(IDClub, IDCat, Name):-
  filter(numChars,Name,"-",2, junkRepo,IDClub, IDCat),
   filter(prohibitedChars,Name,"_*",junkRepo,IDClub, IDCat),
    filter(trim,Name).
```

The first predicate first decomposes and then assembles the given strings. So if there are only two of them it is a clearly first and last name.

Other possibilities aren't so sure, but four pieces most likely means the first two are first name and the other two last name. But what to do with 3 pieces? The most common case is that the first is name and the second two last names, see the listing on page 31. ID 7. If not, one name will be lost.

Second predicate determines the filtering policy and can correct the error provided above.

```
_6 = SMOLCIC-UZUNOVIC
_7 = -
_8 = 1
_9 = 1
_10 = 79
_11 = 4

_6 = BILIC-PAVLINOVIC
_7 = -
_8 = 1
_9 = 1
_10 = 79
_11 = 10

_6 = MIJATOVIC-LUKIC
_7 = -
_8 = 1
_9 = 1
_10 = 79
_11 = 10

_6 = MIJATOVIC-LUKIC
_7 = -
_8 = 1
_9 = 1
_10 = 79
_11 = 13

[78 solutions; 1 inferences;
DVP>
```

Left and right is the very same result with the same content of junkRep structure but with different additional info. Right one has IDs so it is possible with query on Prologs knowledge base to determine full information.

However, there are just 78 cases with dashed name or surname, but mostly surname.

```
Name = SMOLCIC-UZUNOVIC
Problem = -
NumSrch = 1
NumFound = 1
ClID = 79
CatID = 4
ClubName = Vinkovci
CategoryName = M_U18cadets

Name = BILIC-PAVLINOVIC
Problem = -
NumSrch = 1
NumFound = 1
ClID = 79
CatID = 10
ClubName = Vinkovci
CategoryName = U12boys

Name = MIJATOVIC-LUKIC
Problem = -
NumSrch = 1
NumFound = 1
ClID = 79
CatID = 10
ClubName = Vinkovci
CategoryName = U12boys

Name = MIJATOVIC-LUKIC
Problem = -
NumSrch = 1
NumFound = 1
ClID = 79
CatID = 13
ClubName = Vinkovci
CategoryName = U10girls

[78 solutions; 157 inferences
DVP>
```

Filter:

```
filter(nCh,N,"-",1, junkRep).
```

junkRep listing:

```
junkRep(_,_,_,_,_,_).
```

Filter:

```
filter(nCh,N,"-",1,junkRep,IDCl, IDCat),
```

junkRep listing:

```
clubs(IDCl,ClubName),

categories(IDCat,CategoryName,

junkRep(_,_,_,_,IDCl,IDCat).
```

```
junkRepo(OBLAK -,-,1,1,16,6)
junkRepo(- Mihael,-,1,1,32,2)
junkRepo(DEVIC -,-,1,1,82,4)
junkRepo(GRBAC -,-,1,1,82,8)
junkRepo(MUSIC -,-,1,1,82,8)
junkRepo(PETROVIC -,-,1,1,82,8)
junkRepo(- ivan,-,1,1,82,8)
junkRepo(- jelena,-,1,1,82,9)
junkRepo(MUSIC -,-,1,1,82,10)
junkRepo(BARISIC -,-,1,1,57,10)
junkRepo(Jan-Karlo,-,1,1,57,14)
junkRepo(- Mislav,-,1,1,52,2)
junkRepo(- Mihael,-,1,1,52,10)
junkRepo(- Dominik,-,1,1,52,14)
junkRepo(PUPIC -,-,1,1,7,2)
junkRepo(IVICEK -,-,1,1,4,1)
junkRepo(VUJCIC -,-,1,1,3,11)
junkRepo(- Petra,-,1,1,73,5)
junkRepo(BARIC -,-,1,1,8,4)
junkRepo(- MICHEL,-,1,1,53,6)
junkRepo(ANTICIC -,-,1,1,53,12)
junkRepo(- magdalena,-,1,1,49,5)
junkRepo(- danijel,-,1,1,24,0)
junkRepo(----------,-,1,16,11,0)
junkRepo(P_,_*,1,11,0)

junkRepo(__________,_*,12,11,0
```

Filtering is not just removing or beauting existing string. It could remove unwanted but also intentionally replace using provided map, such as example with categories.

Also could add something default or by provided map not just in case of missing but also at some desired places, positions, etc…

So, a possible mistake with the name and surname would be easy to determine. But the junk folder has more problems: incomplete names, incomplete surnames, and some artifacts that are neither names nor surnames.

Incomplete names will be the same in the database. We could not create new information. But we can remove unusable records, so after setting the appropriate filters what we get from junk folders are only 3 irreversible records:

```
DVP> junkRepo(_,_,_,_,_).
_6 = P_
_7 =  *
_8 = 1
_9 = 11
_10 = 0
_

_6 = __________
_7 =  *
_8 = 12
_9 = 11
_10 = 0
_

[2 solutions; 1 inferences;
DVP> junkRepo(_,_,_,_,_,_).
_6 = ----------
_7 = -
_8 = 2
_9 = 16
_10 = 11
_11 = 0
_

[1 solutions; 1 inferences;
DVP>
```

Look at now at this code snippet:

```
Cat2=["M_seniors", "F_seniors", "M_U21juniors", "F_U21juniors",
"M_U18cadets", "F_U18cadets", "M_U16Jr_cadets", "F_U16Jr_cadets",
"U14boys", "U14girls", "U12boys", "U12girls", "U10boys", "U10girls",
"U8boys", "U8girls"], set(noBacktraceArea,5000000),
getCategory(Vc),lowBindingTrue,set(bTL,Vc,Cat2,27,L3),lowBindingFalse.
```

In previous examples above piece of code was in “working thread” and in that situation is not important where on heap is which part of what list.

But that code just prepares L3 list with the intention of using it later. Since its first record is in the lower part of the address space, and all the others in the raised part, after the completion of that preparation, only the first one will remain available. To avoid that situation `lowBindingTrue` directive will ensure everything connected with L3 will be processed in lower space. In contrast to earlier mentioned parser-time binding this one could be called execution-time binding.

Bind type has four scopes: default, global, self and set. Self, once given, does not change in principle. Two types of binding: default and forced. It is used, its underlying engine, as a system tool either.

Consider now this predicate from above code:

```
list(mB,L3,A,AA)
```

It is just a system wrapper around following predicate:

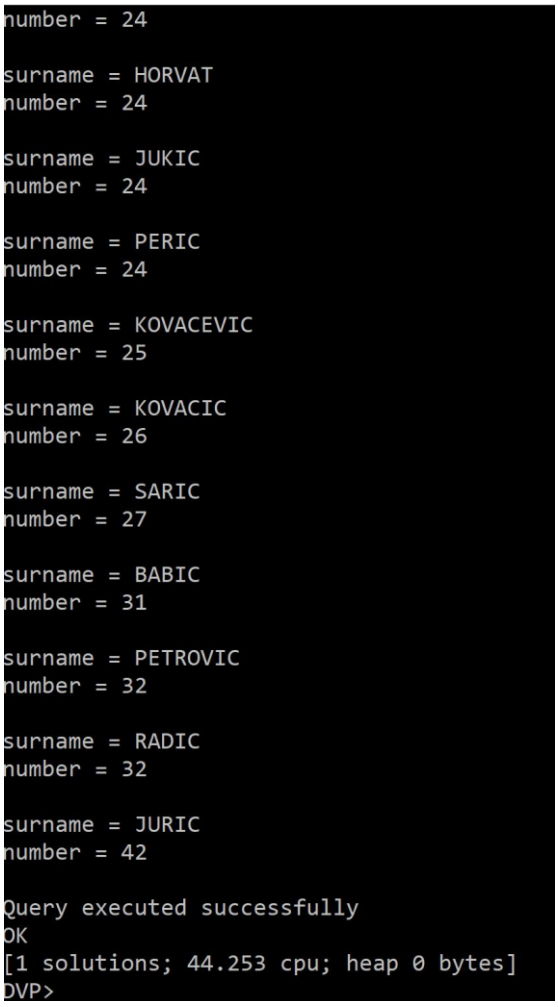
```
matchBind(L3,A,AA):- member(X,L3), X==A, AA=X.
```

Bind type with forced binding will work with variables, thus will compare one of its member with L3 current member, and if match, unify second one with AA, output variable. Since unifying is core system functionality, using it in forced binding must be constrained on as small as possible part of code to avoid unexpected behaviour. System predicate is safe and, in same time, much faster.

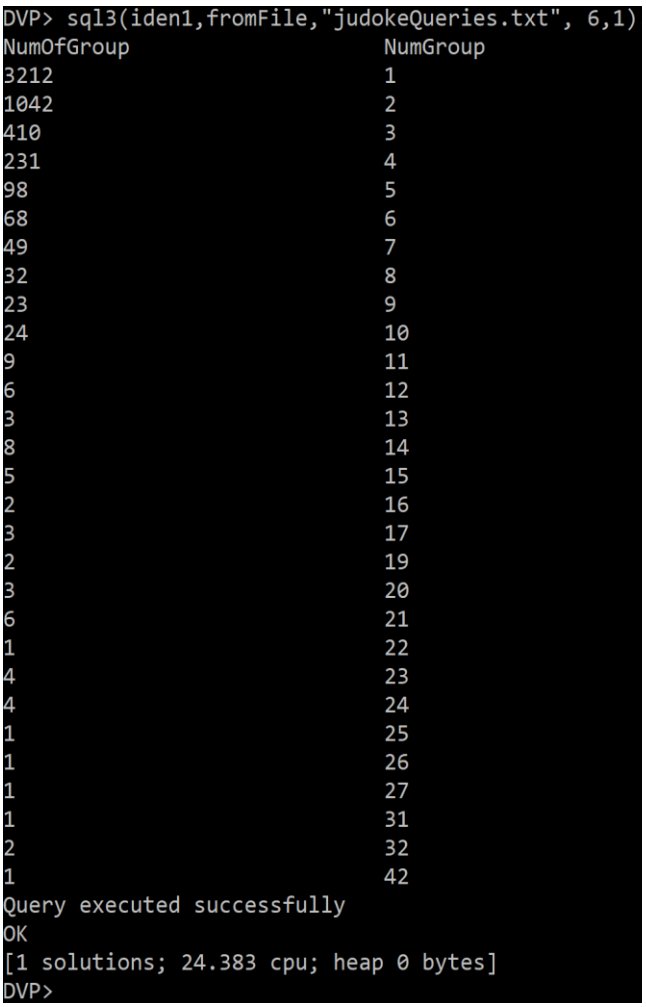
A small introduction to the data analysis

Any kind of analysis capability could be integrated in predicates. Since data analysis is not subject of this paper, just a mention…

Final part of the task, surname analysis, is on pictures below:



```
number = 24

surname = HORVAT
number = 24

surname = JUKIC
number = 24

surname = PERIC
number = 24

surname = KOVACEVIC
number = 25

surname = KOVACIC
number = 26

surname = SARIC
number = 27

surname = BABIC
number = 31

surname = PETROVIC
number = 32

surname = RADIC
number = 32

surname = JURIC
number = 42

Query executed successfully
OK
[1 solutions; 44.253 cpu; heap 0 bytes]
DVP>
```



```
DVP> sql3(iden1,fromFile,"judokeQueries.txt", 6,1)
NumOfGroup                      NumGroup
3212                            1
1042                            2
410                             3
231                             4
98                              5
68                              6
49                              7
32                              8
23                              9
24                              10
9                               11
6                               12
3                               13
8                               14
5                               15
2                               16
3                               17
2                               19
3                               20
6                               21
1                               22
4                               23
4                               24
1                               25
1                               26
1                               27
1                               31
2                               32
1                               42
Query executed successfully
OK
[1 solutions; 24.383 cpu; heap 0 bytes]
DVP>
```

```
SELECT  surname ,count(*) as number
    FROM Members
    group by surname
    order by count(*) asc
```

```
SELECT  count(*) as NumOfGroup,
Z.number as NumGroup  FROM (
    SELECT  surname ,count(*) as number
    FROM Members
    group by surname
    order by count(*) asc) as Z
    group by number
```

Juric is the most common surname, it has 42 members. There are no groups up to 32 members, 10 positions, and then there are two with 32 members: Petrovic and Radic. There is only one gap left, between 27th and 31st place, only 4 positions, a fact which the most numerous Jurić group could qualify as an outlier.

There are 3212 judokas with surname that nobody else in that community has. This is 30,52% and, since this is representative sample for younger population, it has its demographic meaning.

Total number of judokas (SELECT count(surname) FROM Members) is 10523.

Number of judokas with different surname (SELECT count(distinct surname) FROM Members) is 5252. This number is sum of NumOfGroup, left column from right above picture. It is surprisingly close to half of total number: 49,90972%.

Look now at some interestingly looking facts:

```
DVP> set(defaultBind,18,CurrBind),get(mPairs,10524,N1,N2, BindOut,NumPairs).
CurrBind = 18
N1 = [877,1754,2631,3508,5262]
N2 = [12,6,4,3,2]
BindOut = [12-877,6-1754,4-2631,3-3508,2-5262]
NumPairs = 5

[1 solutions; 10.384 cpu; heap 0 bytes; 5 vars ]
DVP>
```

Since we have such a curiosity with the half, let's see what happens to the third. The first number divisible by 3 is one greater than the total, 10524.

One third, 3508 is 109,215% of number of judokas that are in group with just one member.

All unique surnames except those from group 1, 5252-3508=1744 is just 10 less then one sixt of total: 99,42987% of it.

Surnames frequency has demographic and sociological meaning, but that is some other topic…

get(mPairs,ObservedNumber,OutL,OutR,BindOut, NumPairs)

Succeed if calculation of divisors is not longer then MaxCalc time. ObservedNumber is number for which we are calculating divisors. OutL and OutR are lists with divisors at order in which thay are multiplying each other to get ObservedNumber. BindOut is list with binded OutL and OutR. With appropriate defaultBind could be printed in desired order. NumPairs is length of OutL and OutR lists.

First question

It is relatively easy to distinguish between situations in which it can be talked about and others that do not look promising.

Consider version of described predicate:

```
extractList("<div>","</div>", Num, OutList).
```

The OutList will contain all the text on the page, so it would be a good starting point to decide what to do next. In some simple cases it would be necessary to remember only the indexes of the list with the desired data, in others some filtering, rearranging,… must be done.

What if data is not in divs?

```
InputMarkerL=["<div>","<td>","<span>", . . .]

InputMarkerR=["</div>","</td>","</span>", . . .]

extractList(InputMarkerL, InputMarkerR, Num, OutList).
```

Now OutList is list of lists that contain their specific data.

Consider some more sophisticated approach:

```
InputMarkerL=["<td>-<div>","<td>-2<div>","<td>-3<div>-!<span>", . . .]
```

"<td>- " – means that <td> and has something in between that is not important, so jump over it

"<td>-2 " – means that second after <td> is marker

"<td>-3 -!" – third after <td> but disregard

There are number of tools for DOM and HTML that will done this job but almost all of them would need human assistance if source would be even a little bit changed.

Prolog solutions would mostly have the same problem but still at least a shade easier to customize. Especially if the possible change has at least noticeable limits so that the kind of facts or rules of the rules can be inferred.

So there's obviously a lot of talk about structured data, but what about unstructured? Some rare specialized programs or - Prolog as a general practitioner M.D.

Almost all algorithms in this area could be expressed as a dedicated predicate and unlike other approaches, in this case, Prolog's inference mechanism is of great help.

Among other, more or less, standard algorithms, I analyze the specific situation of an incomplete but still usable model. This would be a common situation when trying to create rules for a situation with unsecure or unknown parts. More about that: Incomplete Model.