Joaquín Herrera Ramos A01207504
November 22nd, 2019
Programming Languages: Final project

Excel datasheet to Prolog knowledge base using Java

Joaquín Herrera Ramos A01207504
November 22nd, 2019
Programming Languages: Final project

Context of the problem

Excel is used by an estimated 750 million people worldwide (Cocking, 2017). A lot of information is being stored in millions of books and spreadsheets. This information helps businesses and individuals answer questions about important matters to them. From sales to climate change information. I began this project with an idea of making this data easier to understand and digest.
My first approach was to look around me and find people that used a lot of spreadsheets in their lives. I interviewed a teacher (a director, more specifically) that explained to me that she spends most of her days looking at Excel books and looking for data sometimes hard to get.
I noticed that most of the necessities that this teacher had where about relationships that were contained on the columns and rows of her files.
I came up with an idea: what if she could have a smart application to ask it her questions.
I thought about the paradigms that we have been studying in class and I figured that Prolog could come in handy.
I designed a solution where you could transform an Excel datasheet into a knowledge base accessible by prolog.

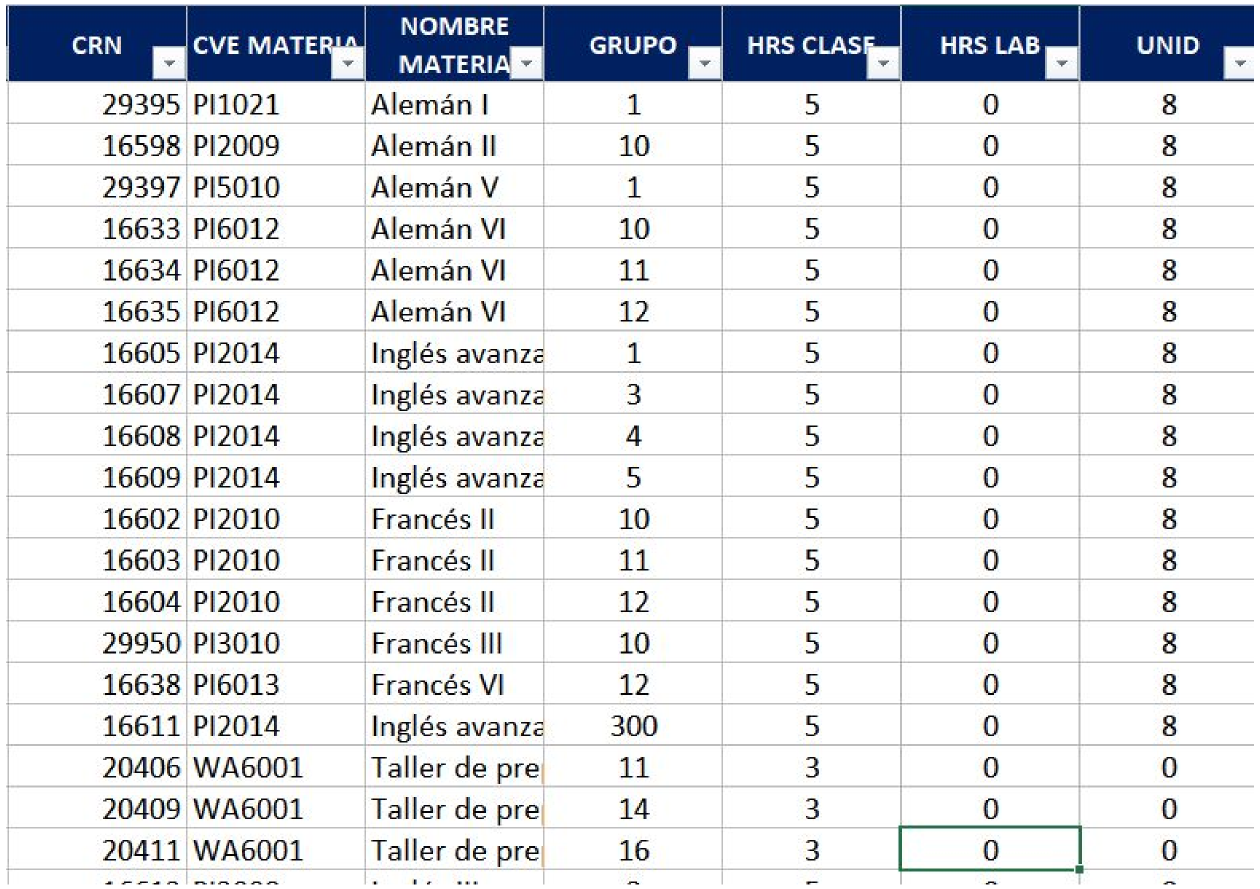
Solution

One of the first problems that I encountered was processing the data from Excel. This is where I decided to use Java. My solution is implemented with Java along with Prolog. I decided to use Java because of its concurrent capabilities. My project involves a lot of string processing to create facts for Prolog to use.
I use an API developed by the Apache Software Foundation called Apache POI. The Apache POI Project's mission is to create and maintain Java APIs for manipulating various file formats based upon the Office Open XML standards (OOXML) and Microsoft's OLE 2 Compound Document format (OLE2). In short, you can read and write MS Excel files using Java. (Apache Software Foundation, 2019).
My Java program creates relationships between a specified identifier or key and any other column, then it prints out these relationships into a Prolog file. I created some Prolog queries to demonstrate the power and capability of having an Excel file translated into Prolog.
My project is stored in a public repository: https://github.com/97joaquinhr/Excel_to_prolog

Results

Inside of the project, there is a folder called “info” that contains the results. At the beginning I had an Excel book that mimics the data that the teacher I interviewed usually uses.

Joaquín Herrera Ramos A01207504
November 22nd, 2019
Programming Languages: Final project



CRN	CVE MATERIA	NOMBRE MATERIA	GRUPO	HRS CLASE	HRS LAB	UNID
29395	PI1021	Alemán I	1	5	0	8
16598	PI2009	Alemán II	10	5	0	8
29397	PI5010	Alemán V	1	5	0	8
16633	PI6012	Alemán VI	10	5	0	8
16634	PI6012	Alemán VI	11	5	0	8
16635	PI6012	Alemán VI	12	5	0	8
16605	PI2014	Inglés avanza	1	5	0	8
16607	PI2014	Inglés avanza	3	5	0	8
16608	PI2014	Inglés avanza	4	5	0	8
16609	PI2014	Inglés avanza	5	5	0	8
16602	PI2010	Francés II	10	5	0	8
16603	PI2010	Francés II	11	5	0	8
16604	PI2010	Francés II	12	5	0	8
29950	PI3010	Francés III	10	5	0	8
16638	PI6013	Francés VI	12	5	0	8
16611	PI2014	Inglés avanza	300	5	0	8
20406	WA6001	Taller de pre	11	3	0	0
20409	WA6001	Taller de pre	14	3	0	0
20411	WA6001	Taller de pre	16	3	0	0

In this example, I used the column CRN to create my relationships since it is a unique identifier. I created, for example, a relationship between CRN and "NOMBRE MATERIA". It ended up looking like this in the prolog file:

```
crn_NombreMateria(29395,"Alemán I").
crn_NombreMateria(16598,"Alemán II").
crn_NombreMateria(29397,"Alemán V").
crn_NombreMateria(16633,"Alemán VI").
crn_NombreMateria(16634,"Alemán VI").
crn_NombreMateria(16635,"Alemán VI").
crn_NombreMateria(16605,"Inglés avanzado II").
crn_NombreMateria(16607,"Inglés avanzado II").
crn_NombreMateria(16608,"Inglés avanzado II").
crn_NombreMateria(16609,"Inglés avanzado II").
crn_NombreMateria(16602,"Francés II").
crn_NombreMateria(16603,"Francés II").
crn_NombreMateria(16604,"Francés II").
crn_NombreMateria(29950,"Francés III").
crn_NombreMateria(16638,"Francés VI").
crn_NombreMateria(16611,"Inglés avanzado II").
crn_NombreMateria(20406,"Taller de preparación para examen institucional de
inglés").
crn_NombreMateria(20409,"Taller de preparación para examen institucional de
inglés").
crn_NombreMateria(20411,"Taller de preparación para examen institucional de
inglés").
crn_NombreMateria(16613,"Inglés III").
crn_NombreMateria(16657,"Lengua española, arte y literatura").
crn_NombreMateria(16658,"Lengua española, arte y literatura").
crn_NombreMateria(16646,"Lengua española, arte y literatura").
crn_NombreMateria(20404,"Taller de preparación EXANI II").
crn_NombreMateria(29398,"Literatura I").
```

I later implemented a query to work with this relationship, this is located in the queries.pl file:

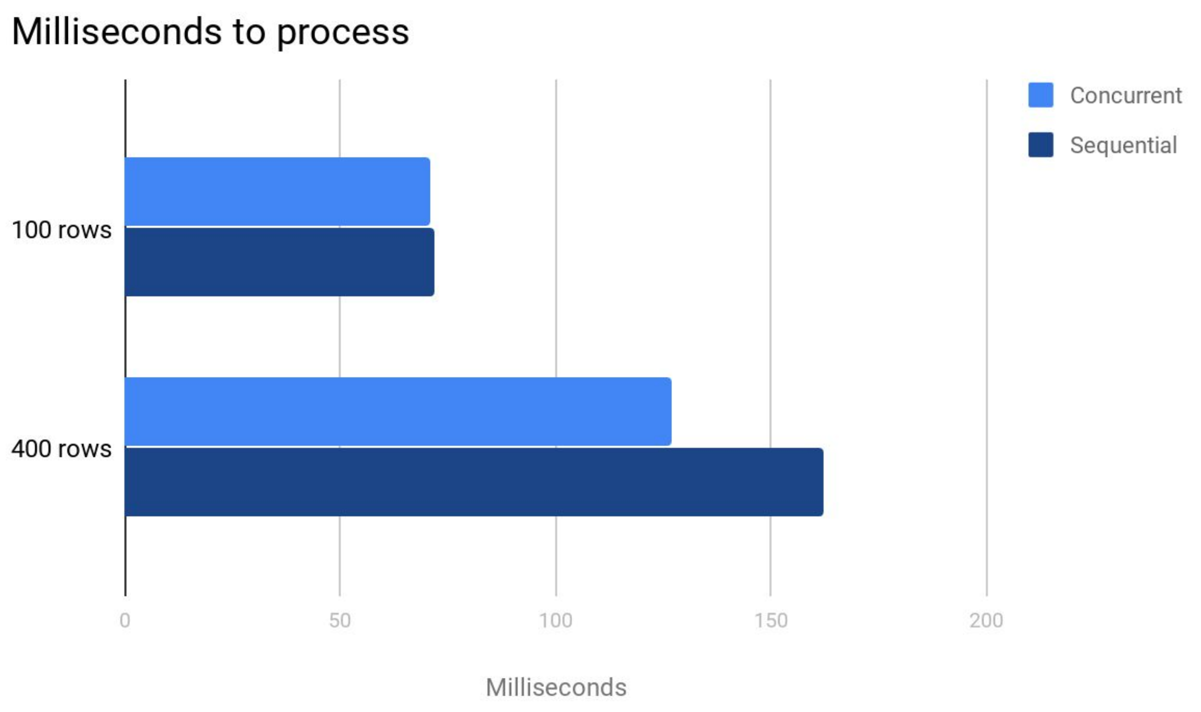
```
cursosH([],X,X):-!.

cursosH([H|T],X,Res):-
    crn_NombreMateria(H,Curso),
    appendM(X,[Curso],NewX),
    cursosH(T,NewX,Res).

cursos(Nomina,X):-
    findall(CRN,crn_Nómina(CRN,Nomina),Aux),
    cursosH(Aux,[],X).
```

With this queries you can get a list of all the course names that a teacher gives. This is useful for the teacher that I interviewed since she usually needs to identify quickly how many classes a teacher gives and which courses are these.

I found that by using RecursiveAction in Java, the concurrent string handling became much more efficient, this resulted more significant for larger Excel books.
For example, I ran several tests of concurrency for a 100-rows book and my average time of processing for the concurrent implementation was 71 milliseconds, and sequential was 72 milliseconds. Which is almost the same. But as I increment the size of the spreadsheet (400 rows), things change: the average time for concurrent becomes 127 milliseconds and the average time for sequential turns into 162 milliseconds.



The Prolog file ended up containing more than 700 relationships, this is why concurrent processing made sense.

Conclusions

After I finished the first version of my project, I showed the teacher the results of my work and she found my project to have a lot of potential for her daily tasks. As of right now, my project still lacks a friendly user experience but the analysis capabilities is already present.
If I were to continue working on this project, I would even think about using a voice interface. I think it could come handy to ask literally what you want to know from an Excel file. With some intent processing and language understanding, I would be able to query my prolog knowledge base. The implications of a project like this would be huge because as I explained earlier, a lot of people rely on Excel to store information. We just need a more accessible way of understanding it.

Joaquín Herrera Ramos A01207504
November 22nd, 2019
Programming Languages: Final project

Setup instructions

My file structure looks like this:

- info/
    - Proyecto.xlsx
    - data.pl
- java/
    - excel/
        - pom.xml
    - src/
        - Main.java
        - Process.java
- queries.pl
- testInput.txt

Inside of the info folder, there is an example of what kind of Excel books my program accepts and an example of output in data.pl.
Inside of the java folder you can find my source code, which is composed of 2 files: Main and Process. Main defines several methods that make possible the reading and writing parts of the project and Process handles the concurrent processing of strings to make the relationships.
Inside of java/excel/ there is a file called pom.xml. A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project (Apache Maven Project, 2019). In this file I define "org.apache.poi" as a dependency for reading an Excel file. There are other dependencies defined as well.
The setup instructions consist of the following:

1. Run the program and follow the menu:
2. Provide the excel file path.
3. Provide the PL file path.
4. Provide how many IDs you want to process and repeat the following for each:
    a. Provide the row that contains the unique identifiers.
    b. Provide the name of the unique identifiers.
    c. Provide the number of rows to read for this ID.
    d. Provide the number of each row to read (as many as the number provided in the step above).
5. Wait for results.
6. Check the results on the path of the PL file provided.

I created some rules inside of my queries.pl file to demonstrate how to use a separate prolog knowledge base.
Note: my documentation is commented inside of the java project.

Joaquín Herrera Ramos A01207504
November 22nd, 2019
Programming Languages: Final project

References

Apache Software Foundation. (2019). Apache POI - the Java API for Microsoft Documents. *Apache POI.* Retrieved from https://poi.apache.org/

Apache Maven Project. (2019). Introduction to the POM. *Maven.* Retrieved from https://maven.apache.org/guides/introduction/introduction-to-the-pom.html

Cocking, S. (2017, December 13th). Seven reasons why Excel is still used by half a billion people worldwide. *Irish tech news.* Retrieved from https://irishtechnews.ie/seven-reasons-why-excel-is-still-used-by-half-a-billion-people-worldwide/