Université de Lorraine **I**DMC **A**TILF

MSC Natural Language Processing

UE 805 – Supervised Project

Realization Report

Database Creation For The Lex.E.M Project

Academic Year: 2020-2021

Host Organization: ATILF

Students:

Morgan RUIZ-HUIDOBRO

Soklay HENG

Supervisors:

Etienne PETITJEAN

Marie Laurence KNITTEL

Samantha RUVOLETTO

Reviewer:

Bruno Guillaume

June 18, 2021

Contents

Abstract **2**

Introduction **3**

1 Presentation Of The Subject **4**

1.1 Relational Database 4

1.2 Corpus 5

1.2.1 Henkeler 5

1.2.2 Kern-French 5

1.2.3 Lyon 6

1.2.4 Standford-French 6

1.2.5 CoLaJE 6

1.2.6 JEMS 6

1.3 Data Extraction 6

1.4 NLP Task 7

1.5 SQL 8

2 Work Done **9**

2.1 Pipeline 9

2.2 Data In Corpus 10

2.3 Extraction Of Data From XML 10

2.3.1 First Function 11

2.3.2 Second Function 11

2.3.3 Third Function 11

2.4 Issue Caused By NLP Tool 12

2.4.1 1st Alternative 12

2.4.2 2nd Alternative 12

2.4.3 Alternative Chosen For The Database 13

2.5 Schema Of Database 13

2.6 Database With SQL 14

2.7 Result 15

3 Discussion **17**

4 Conclusion **17**

Annexes **19**

Bibliography **28**

Abstract

The main objective of the Lex.E.M project and literature review have been introduced in the bibliography report of the first semester. We then continued our work by getting deep down into technical part and made the concept become reality. This report will resume the work realized by us during our participation to the Lex.E.M project. Technical steps will be introduced in this realization report. Our role was to create a database containing data on the language production of children of age 2 to 3. The database was created from 6 different corpora, 4 from CHILDES , one from Colaje and the JEMS corpus for the last one. The goal of the Lex.E.M project with this database is to develop remediation tools for children with language-related difficulties.

Introduction

The supervised project consists in the elaboration of a database containing children's real language production for the Lex.E.M[1] project.

The Lex.E.M[1] project is realized in collaboration with kindergarten from the REP[2]. The need for the project comes from an observation by teachers; lexical deficiencies have been observed for some children in comprehension and production. This can be explained by the fact that those schools have children with really different profiles and numerous allophone children with or without knowledge of French.The end goal of the project is to create remediation tools that support lexicon access for children between age 2 and 3. Having a new corpus containing language production of children in school was a necessity. Before the creation of the JEMS corpus by the Lex.E.M project, no other data were available on the language production in school context for French-speaking children. The recording took place in different school from non REP area.

This supervised project represents the second step of the lex.E.M project, meaning the creation of a database. The database we had to create intended to represent most accurately the language production of children. Therefore, it needed to contain information related to phonology, morphology, and lexical information.

This report will illustrate the process of creation of such database. Consequently, we will discuss topics including relational database, extraction of data, linguistics tools and SQL. Firstly, we will present the different topics we have encountered during the project, and we will talk about their applications in our work. Then, we will present the different steps we took to create the database containing the necessary information we wanted to have in the database.

[1] lexicon for kindergarten

[2] reseaux d'éducation prioritaire

1 Presentation Of The Subject

This section will briefly present the topics seen during the development of the database. For the sake of your understanding, we will introduce some relevant concepts. That way you will be familiar with them when we will talk about the work we have done during the supervised project.

1.1 Relational Database

The first subject we need to address is the creation of a relational database[2] which is the main goal of this supervised project.

Databases are sets of data that have been structured, for example into multiple tables where columns hold the attributes of the data. Relational database means that some data attributes are related to one another. A database enables us to find information more easily by the structure itself. Getting access to the data of a database is facilitated by the tables. We, therefore, can query[3] a special element of our data. We will discuss more in details about information retrieval in the SQL[4] part of our report.

A table contains multiple columns. These columns can contain different keys for each table. A primary key is generally used in tables; it is employed to create a unique value, and this value can't be null. When creating/inserting a new row to the table, this unique value is attributed. That way when the system accesses the table, it will use this primary key to access the data stored. A foreign key is used to link the data between one table to another. Usually, the foreign key of one table refers to the primary key of another table.

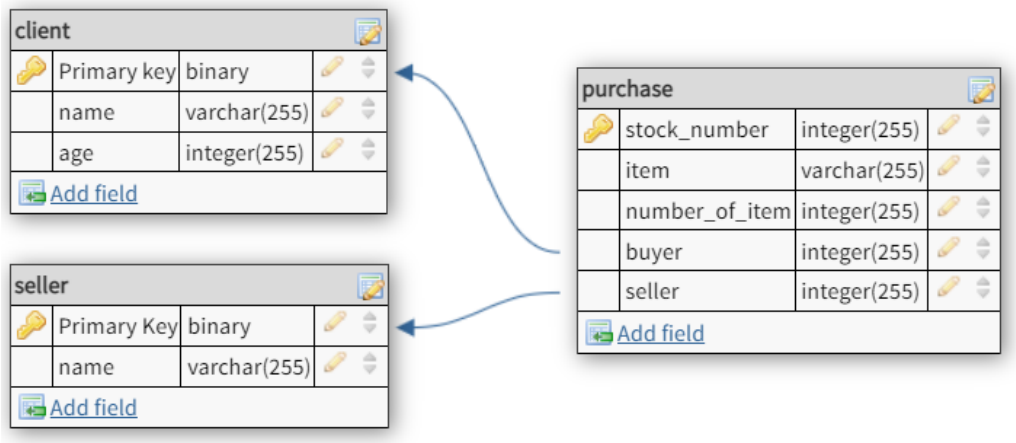

Figure 1: Example Of Relational Database

[3] request for data from a database table

[4] structured query language

One example has been provided to illustrate this, as can be seen in the figure [1], the table “purchase” is linked to two tables thanks to the foreign keys named “buyer” and “seller”. The key “seller” known as foreign key in the “purchase” table is linked to the primary key of the table “seller”. The key “buyer” known as foreign key in the “purchase” table is linked to the primary key of the table “client”. Each table contains different information. With this sort of database, we can find the purchase made by certain client, or the sale done by a certain seller. We can even find what item people over a certain age purchase. This sort of relational database could allow market study.

1.2 Corpus

The definition of a linguistic corpus we can give for our project is a collection of language production.

Our database contains data from 6 different corpora. From the CHILDES[3] dataset, we have chosen four datasets named Hunkeler, Kern-French, Lyon and Standford-French. We will also include the CoLaJE[5] and JEMS dataset. All the corpora collected have already been annotated and come in XML format. Some of the corpora selected are available in free access on the ortolang[4] website.

1.2.1 Henkeler

The Henkeler[5] dataset comes from a study realized by Hervé Hunkeler. This corpus is the result of research on the development of early lexicon. He followed the development of dizygotic twins, Camille and Pierre. He recorded their interaction with their mother when the children were in between age 1,6 and 2,6.

1.2.2 Kern-French

The Kern-French[6] dataset compiles the language production of four children Baptiste, Emma, Esteban and Jules. The data was collected every two weeks and consists of 1 hour of spontaneous speech from children whose ages are between 0,8 and 2,1 years old. The recording took place in the children’s home. The data was phonetically transcribed in API[6].

[5] communication langagière chez le jeune enfant (language communication of young children)

[6] International Phonetic Alphabet

1.2.3 Lyon

The Lyon[7] corpus contains audio/video recording of five French-speaking children from age 1 to 3. In our database we will use the data collected from Anaïs, Marie, Nathan and Theotime. The recording took place in the children's home, producing spontaneous-speech. Each child was recorded for 1 hour every 2 weeks. The corpus has been transcribed in API. The goal of the project was to study early phonological and morphological development.

1.2.4 Standford-French

The Standford-French[8] contains speech production of six children from age 0,9 to 1,7. This database has been used to analyse phonetic tendencies in the early stage of language development.

1.2.5 CoLaJE

The CoLaJE[9] corpus consists of audio/video recording. The children were filmed 1 hour each month. The data is phonetically transcribed. The goal of the project was to build a database with children's production from birth to age 7. The aim was to collect new french data, to improve the transcription system, and to study the development of grammatical tools used by children. The objective was also to analyse the data to find regularities in acquisition for each child and also across children.

1.2.6 JEMS

The JEMS[1] corpus contains recording of speech production of children in a standard school environment. The data was transcribed in API. The goal of the corpus was to provide information on the language used by children in school. In the long run, the data will enable the creation of remediation tools to help children with language-related difficulties.

1.3 Data Extraction

Extracting the necessary contents to include into our database is also a crucial step for the supervised project. The different corpora utilised stocked the data in XML files. XML is a markup language[10]. It stores information through the use of tags, and the document is structured thanks to that. Throughout our NLP master's course, we have been introduced to a Python library named "Beautiful Soup"[11] to parse and retrieve

data from XML file. The data can be extracted from XML files by other methods, but we will focus on using "Beautiful Soup" during our project. The "Beautiful Soup" library offers a XML parser. Once we parse the document, we can navigate the parse tree with the function find() and findall() of the module. The tree of our XML file looked like the schema [9] in annex. The branch "participant" gives information about the different people speaking. The branch "u" contains the data related to speech with sub-branches called "orthography", "ipaTier" and "groupTier". From the branch "u", we can extract lexical, phonological and morphological information.

1.4 NLP Task

During our supervised project, we have also gone through different NLP tasks, such as tokenization[7], Part-of-speech tagging[8], lemmatization[9] and regular expression[10].

The tokenization, POS-tagging and lemmatization were handled by "TreeTagger"[12]. It is a system developed by Helmut Schmid in the TC project at the Institute for Computational Linguistics of the University of Stuttgart. We can use TreeTagger in a Python environment thanks to the Python library "treetaggerwrapper"[13]. This tool comes with a function taking as input text and returning part-of-speech tag and the lemma for each token of a sentence.It supports many languages, including French which is the language we are working on and and is adaptable to other languages if there is an availability of a lexicon and a manually tagged training corpus for those languages.

The regular expression was handled by the Python module "re"[14]. We used it when we parsed the XML file. We needed regular expression when we extracted the dates (date of recording, age ,birthday) and to change their format.

[7]tokenization: separates piece of text into smaller units called tokens. A token can be a word, characters or sub-words.

[8]Part-of-speech tagging: marks a word in a text with the corresponding part-of-speech information (based on the definition and context). It is linking a word to its grammatical category, such as noun, verb, adjective, adverb, etc.

[9]lemmatization: marks a word in a text with its lemma. A lemma is the base form of a word, a word without trace of inflexion. We can form many related words from lemma.

[10]regular expression: sequence of symbol and character where a pattern can be found and searched within another string.

1.5 SQL

Structured Query Language known as "SQL" is a standard language used to store, manipulate and retrieve data in databases[15]. SQL is used to communicate with a database.

For this supervised project, we decided to work with "MySQL" to create the database although there exists many database systems to manage database, such as SQL Server, MS Access, Oracle, Sybase, Informix, Postgres, etc. This is due to the fact that MySQL is fast and is an easy-to-use relational database management system based on SQL. In addition to this, it is open-source and free to download.

To interact with the databases, multiple SQL[16] statements are used. 'SELECT' allows the extraction of data from a database and from specific table. 'UPDATE' manipulates the data already present into the database; it is updating the data already exist there. 'DELETE' also manipulates data from the database; it deletes data from a database. 'INSERT INTO' is a statement used to insert new data into a database. 'CREATE DATABASE' allows the user to create a new database. 'ALTER DATABASE' modifies a database already created. 'CREATE TABLE' is useful to create new tables into the database. 'ALTER TABLE' modifies a table already existing. DROP TABLE deletes a table of a database.

2 Work Done

This section will illustrate the different steps we took along the project. It will thoroughly describe how each task was achieved.

2.1 Pipeline

Below is the pipeline from the data extraction from XML files to the final result.

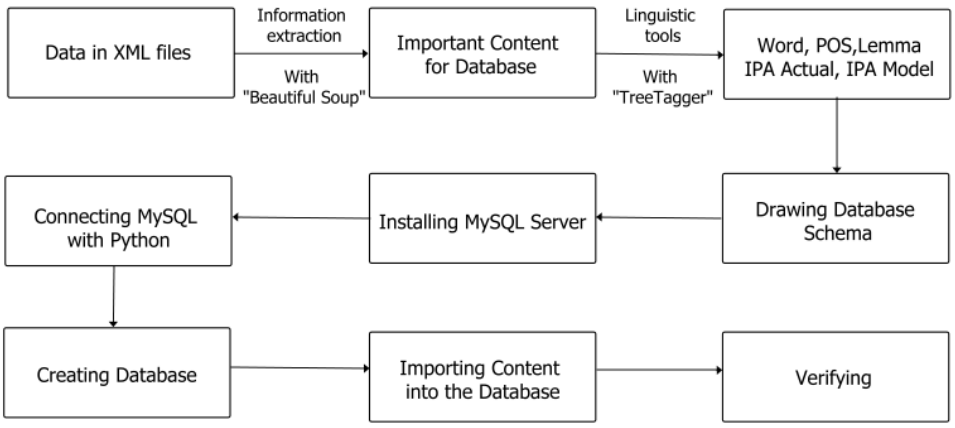

Figure 2: Pipeline

In order to keep track on our progress during the realization part of the LeX.E.M project and to complete it successfully, we have drawn a pipeline with the following steps. The first step that has been taken into account is the retrieval of data that will be used later in the implementation part. This step involves extracting important information from XML files, using "Beautiful Soup" Python module. Following that, we utilized the linguistic tool named "TreeTagger" to do part of speech tagging and lemmatization automatically. Once the data pre-processing part was completed, the next important step was to design a database schema. The database needed to be structured in a certain way to include all the information coming from the XML files. Following this, we needed to install MySQL and establish a connection between the server and Python. Then, the next step was to create a database and to import data into the database. Finally, we had to verify the result by making database queries to ensure that the database really contained the information we wanted to implement and was designed correctly.

More details on how each step was achieved will be illustrated in the later section of this report.

2.2 Data In Corpus

All the corpora have the same structure, but it doesn't mean they contain the same information. The following table recapitulates the information contained in the XML files for each collection of data.

	ipa-actual	ipa-model	morphology
CHILDES-KernFrench	yes	no	no
CHILDES-Lyon	yes, for children speaker	Yes, for children speaker	yes
CHILDES-Henkeler	Yes, for children speaker	no	yes
CHILDES-Standford	yes	no	no
CoLaJE	Yes, for children speaker	Yes, for children speaker	Yes, for children speaker
JEMS	Yes, for children speaker	Yes, for children speaker	no

Figure 3: Corpus Content

The kernFrench and Standford collections of language production contain only the language production of children but not the production of their interlocutor. The other collections recorded the interaction between the participants. It is also important to be aware that lexical units are sometimes annotated as unintelligible. Nonetheless, an occurrence can have an undefined lexical unit but still has phonological information.

2.3 Extraction Of Data From XML

All the corpora are presented in the same format. They are composed of XML files with the same root tree (see figure [9] in annex).

We have extracted the data to insert into the database successfully thanks to the Python module named "Beautiful Soup". Beautiful Soup is a Python library which is helpful for data retrieval from HTML and XML file format. Beautiful Soup can be installed by running the following command in the terminal.

```
pip install beautifulsoup4
```

Once the module has been successfully installed in the terminal, we can use the function by importing the module to our Python environment[11] using the following command.

```
from bs4 import BeautifulSoup
```

Three functions were created to extract data in the most efficient way. The extracted data was stored in a pandas DataFrame. Pandas is a Python library used to handle data[17]. Pandas can be used to plot data and is also useful to display data in tables. Pandas can be installed by running the following command in the terminal.

[11] python environment: we used Jupyter Notebook

```
pip install pandas
```

Once the module has been successfully installed in the terminal, we can use the function by importing the module to our Python environment using the following command.

```
import pandas as pd
```

We will now describe all the extraction function created one by one in the following subsections.

2.3.1 First Function

The first function we created was to extract information on the XML file itself. It returns a pandas DataFrame with the columns 'file_name', 'content' and 'link'. They respectively refer to the name of the file, the whole content of the file and the link where we can find the database. All the data in the table are of string datatype. The commented function can be found in annex (code [1]).

2.3.2 Second Function

The second function retrieves data on the different participants speaking in the file. It returns a pandas DataFrame with columns named 'id', 'role', and 'age'. They respectively refer to the id given to the participant in the file, the role of the participant (target child, mother, teacher...) and the age of the participant at the time of the recording. All the data in the table are of string datatype. The format of the age extracted was changed to a format "yyyy/mm/dd". The commented function can be found in annex (code [2]). The function had to be adjusted slightly for the Standford-French collection. The date of recording and the date of birth were noted differently compared to other collection. So, the only modification we needed to make was in the regular expression that searches the pattern of the dates.

2.3.3 Third Function

The third function retrieves data related to the sentences. This function returns two different pandas DataFrames.

The first DataFrame contains three columns named, 'sentence_id', 'speaker' and 'sentence'. They refer respectively to the id given to a sentence in the file, the id of the participant speaking, and the sentence uttered.

The second DataFrame contains nine columns named, 'sentence_id', 'speaker', 'sentence', 'word', 'ipa-actual', 'ipa-model', 'POS', 'lemma', and 'morphology'. The first three

columns contain the same type of information as those in the first DataFrame. The other columns refer to:

- Word = token of a sentence.
- IPA Actual = transcription in IPA of the actual phonological production of a participant.
- IPA Model = transcription in IPA of the ideal phonological production of the token.
- PoS = Part-of-speech tag of the token.
- Lemma = lemma of the word token.
- Morphology = morphological information on the token.

The commented function can be found in annex (code[3]).

2.4 Issue Caused By NLP Tool

The tokenization used by TreeTagger seemed to be an issue for the database creation because it didn't correspond to the tokenization used in the XML file. Therefore, with this issue, two alternatives were available for us.

2.4.1 1st Alternative

We keep the tokenization by TreeTagger to realize PoS tagging and lemmatization. This method required us to separate the data into 2 tables. One table contains word, pos and lemma, and another one contains word, ipa-actual, ipa-model and morphology. The PoS and lemma cannot be aligned with the phonological and morphological information since the tokens are not the same. This method presents constraint for the future use of the database. For example, we will not be able to study the phonological aspect of language alongside the grammatical features.

2.4.2 2nd Alternative

We keep the tokenization used in the XML file to realize PoS tagging and lemmatization. This method allows us to create only one table to store data on occurrences. However it also presents a drawback. The token selected in the XML file are sometimes not well supported by TreeTagger. This causes a token to be assigned with the wrong PoS or lemma value. For example, the token "s'appelle" will have as PoS "NOM" and as

lemma “s’appelle”. Since most of the utterances have been tokenized with token supported by TreeTagger, this type of error doesn’t represent the majority. This method presents an advantage for the future use of the database despite the error because the morphological and phonetic data can be studied alongside grammatical data.

2.4.3 Alternative Chosen For The Database

We considered the goal of the Lex.E.M project to choose an alternative over another. The aim is to study the language production of children to develop in the long term remediation tools for children with language difficulties. Therefore, it seems to be more logical to keep the system that allows to analyse in parallel different linguistic characteristics. We, therefore, decided to create our database (and by our consequent data extraction) based on the second alternative.

2.5 Schema Of Database

In order to build our database efficiently, we need to make a draft about what information had to be included in our database and what kind of relationship one table should have with other tables. Consequently, we have designed a database schema shown below. We have identified the datatype of each column and the relation of tables with other tables.

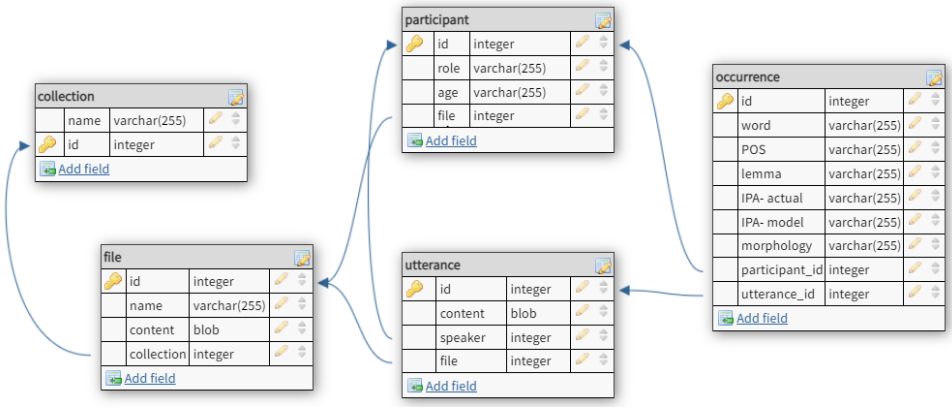

Figure 4: Lex.E.M Database Schema

Our database is made up of five tables named collection, file, participant, utterance and occurrence. All the table possess a primary key named id, and some of them have foreign keys defining the relation between tables.

The table file has a foreign key linking it to the collection table. It allows us to see which file belongs to which collection. The table participant and utterance have a

foreign key making the link with the file table. It is a way to know which participant was speaking in a file and which sentence was uttered in which file. Furthermore, the table utterance is also associated with the table participant by another foreign key. It is linking the utterance with the participant who enunciated it. The table occurrence is linked by two foreign keys to the tables participant and utterance.

The linguistic content we included in our database can be illustrated as follow:

- PoS: indicates part of speech of each lexeme.
- Lemma: indicates the base form of the lexeme.
- IPA Model: indicates the expected pronunciation of a lexeme, noted in IPA.
- IPA Actual: indicates the phonological production of a lexeme realized by a participant and noted in IPA.
- Morphology: indicates morphological annotation of each lexeme.

2.6 Database With SQL

To create the database, we had to go through multiple steps. Once MySQL server had been installed successfully, we used Python library "mysql.connector" to use SQL in Python. This Python module can be installed by running the following command in the terminal.

```
pip install mysql-connector-python
```

Once the module has been successfully installed in the terminal, we can use the function by importing the module to our Python environment using the following command.

```
import mysql.connector
```

When the module has been imported to Jupyter Notebook without any error, it indicates that the module is ready to use. Before we created the database, we need to create a connection between MySQL and Python. We created a function taking as parameters the input needed to connect to the server and the name we wanted to give to our database. The code commented can be found in annex [4].

We then implemented the table presented above with the SQL statement 'CREATE TABLE'. The commented code can be found in annex [5].

After that, we inserted the name of the corpora into the table collection with the function "insertcollection()". The commented code can be found in annex [6].

Following this, we had to create a function to insert data into the database. For each file, the data needed to be inserted at the same time for efficiency reason. Because FOREIGN KEY links the tables together, it was more efficient to insert data file by file to find more rapidly the foreign key inserted previously into the other tables. We also needed to modify the data type to insert it into the database because SQL doesn't support some particular data types. We encountered issue with the CoLaJE dataset while we were querying the id to insert as foreign key value. To collect the id, we were using a SELECT statement, and the server keeps closing before finding the value. We assume the error came from the fact that we were processing to many occurrences by file. To resolve it, we increased the query limit with the following command.

```
mycursor.execute('set GLOBAL max_allowed_packet=67108864')
```

Our method to create the function inserting data was to insert table by table for each file. Firstly, we inserted data into the table file, then participant , after that utterance, and we ended with occurrence. The 3 functions retrieving data described earlier need to be called inside the function feeding the data. The three functions take as parameter the path to a file. We decided to create a JSON dictionary containing as key the name of the collection and as values the path of the files. We used JSON to store the path to feed it later more easily to the different functions. The annotated code can be found in annex [7].

2.7 Result

In this section, we will present some examples of data stored in the database. For each collection, we will give example of some tables. The following table presents the example of data extracted from the table occurrence for the Kern-French collection. As can be seen and as we described before, the Kern-French collection contains mostly phonological data. The notation 'yyy' for word means that annotators of the corpora weren't able to decipher a proper lexeme.

id	word	ipa-model	ipa-actual	POS	lemma	morphology	participant_id	utterance_id
739861	yyy	None	dada	None	None	None	1069	220787
739862	yyy	None	dœ:	None	None	None	1069	220788
739863	yyy	None	œ:	None	None	None	1069	220789
739864	yyy	None	dœ	None	None	None	1069	220790
739865	yyy	None	da	None	None	None	1069	220791

Figure 5: Extract Of Occurrence Table For The Kern-French Collection

In the next figure, we present an extract of the table participant for the collection Lyon.

id	role	age	file_id
1	Target Child	1/0/23	1
2	Mother	25/0/0	1
3	Investigator	25/0/0	1
4	Target Child	1/0/23	2
5	Mother	25/0/0	2
6	Investigator	25/0/0	2

Figure 6: Extract Of Participant Table For The Lyon Collection

The next table illustrates the data we can find in the table utterance for the Henkeler collection.

id	content	speaker	sentence_id	file_id
214960	b"et l\xc3\xa0 c'est quoi l\xc3\xa0"	976	4af8c8ee-bb28-4ff2-85e4-e4e878f0a541	328
214961	b'attends la page est coll\xc3\xa9e'	976	d01821b9-8ea0-4633-b081-98c6a378c887	328
214962	b'oh un dauphin'	976	7e556c1e-250d-4340-8b03-5b73a5c4d084	328
214963	b'comment ils font les poissons Camille'	976	56f8d9ec-7f20-44da-83ff-65a99cbb9634	328
214964	b'xxx'	975	419dca90-5fb7-4bac-a111-e2a2a2d39381	328
214965	b"ouais et l\xc3\xa0 c'est quoi \xc3\xa7a"	976	2ba90793-d846-4cb0-bd00-1859cb51288c	328
214966	b'comment il fait le chien'	976	a1e916cc-3d90-46f0-9e21-185ade318755	328
214967	b'comment il fait le chien'	976	71924ba1-0ca1-4d9c-beac-17a8a4ea4263	328
214968	b'wouf'	976	f0c7f116-c481-4aba-a1e8-9b809e278cc7	328
214969	b'wouf'	976	dbcec842-3561-4e9b-ad24-982571ab0f62	328
214970	b'xxx'	975	1d21a370-bb53-490f-bfc1-fdcfa137940a	328

Figure 7: Extract Of The Utterance Table For The Henkeler Collection

Once the data were completely implemented to the database, we could count 6 collections, 646 files, 399149 utterances and 1304750 occurrences.

collection	KernFrench	Lyon	Henkeler	Standford	CoLaJE	JEMS
number of files	129	327	47	34	80	29
number of utterances	50106	214959	5827	5166	116159	6935
average number of utterances by file	388	657	124	151	1452	239
number of occurrences	59202	720848	19012	10332	463926	31430
average number of occurrences by file	459	2204	404	304	5799	1083

Figure 8: Database Size

3 Discussion

The result obtained can be improved by different systems. For example, the missing data in IPA-model could be automatically filled by a transcription system taking as input the word. Obviously, some words noted as unintelligible could not be transcribed even with this sort of system.

Moreover, the alternative chosen to create the database added wrong value for PoS and lemma. A manual verification could be applied on the data. This method would be extremely time-consuming if we take into consideration the size of the database. If other corpora were to be added in the future, it could be a good idea to tokenize the sentence with the same system used for PoS-tagging and lemmatization before any transcription.

Regarding another remark on the code we have realized to insert data into the database, the function we have created takes a long time to process all the file contained into a collection. A limitation in our abilities have hindered the creation of a faster system.

4 Conclusion

To sum up, we have successfully completed this project with satisfying result by creating a database storing information about children real language productions, which was the main objective of this supervised project. This database will be beneficial for later usages with different purposes as the database we have created will serve to create an available web interface. For the Lex.E.M project, the database will be exploited to help children with language-related difficulty. The database can be used to develop tools or applications that will be beneficial for children facing lexical problems. Regarding our skills to complete this project, we have benefited from the introduction course of data parsing using "Beautiful Soup" Python module in our first semester of our NLP master's course. The course on SQL we have taken during the master was also useful to understand database query. However, we also needed to explore it more and learn new skills. Complete this project required skills such as creating relational database with MySQL and knowledge on the different functionality of TreeTagger. We believe that the new skills we have learnt from this project will be beneficial for our second year of the master's course and for our future career path.

Acknowledgment

We would like to thank our supervisors Etienne Petitjean, Marie Laurence Knittel and Samantha Ruvoletto for their support and guidance during this project. We would also like to thank them for the chance we had to work alongside them on the Lex.E.M project. We also want to express our gratitude to Maeva Sillaire, a student in sciences of language, who followed the project with us and helped us during the process. We also want to express our appreciation for the opportunity, given by the master in Natural Language Processing, to work on this initiative and interesting project as part of the curriculum.

Annexes


<?xml version="1.0" encoding="UTF-8" standalone="true"?>
<session version="PB1.2" corpus="Yamaguchi" id="010410" xmlns="http://phon.ling.mun.ca/ns/phonbank">
<header>
<date>2006-05-08</date>
<language>fra</language>
<media>010410.mov</media>
</header>
<participants>
<participant id="CHI">
<role>Target Child</role>
<name>Adrien</name>
<sex>male</sex>
<birthday>2004-12-28</birthday>
<age>P1Y4M10DT0H0M0S</age>
<language>fra</language>
</participant>
<participant id="MOT">
<participant id="INT">
<participant id="FAT">
</participants>
<transcribers/>
<userTiers>
<tierOrder>
<transcript>
<comment type="Code">pid 11312/c-00043006-1</comment>
<comment type="Date">08-MAY-2006</comment>
<u id="d416b504-9a52-458d-9607-14be29a06604" excludeFromSearches="false" speaker="MOT">
<orthography>
<g>
<w>Adrien</w>
?
</g>
</orthography>
<ipaTier form="model">
<pg>
<w/>
<sb/>
</pg>
</ipaTier>
<ipaTier form="actual">
<pg>
<w/>
<sb/>
</pg>
</ipaTier>
<alignment type="segmental">
<ag length="0"/>
</alignment>

<flatTier tierName="gestes">El arrive dans la pièce .</flatTier>
<flatTier tierName="situation">A est déjà assis dans sa chaise haute .</flatTier>
<groupTier tierName="Morphology">
<tg>
<w>n:prop|Adrien</w>
<w>?</w>
</tg>
</groupTier>
</u>

Figure 9: XML Tree Example

```
def file_(file_path):
    #read the xml file
    infile = open(file_path,"r", encoding="utf8")
    contents = infile.read()
    soup = BeautifulSoup(contents,'xml')
    # branch in the file where you can find information on the file
    doc = soup.find_all('session')
    # create a dictionary and a panda DataFrame to stock data later
    dico3 = {'file_name':'','content':'', 'link':''}
    k = pd.DataFrame(columns = dico3.keys())
    #retrieve the data
    for i in doc:
        id1 = i['id']
        link = i['xmlns']
        #dictionary with the implemented data
        dico3 = {'file_name':id1, 'content':contents,'link':link}
        #add data to the DataFrame thank to a dictionary
        k = k.append(dico3, ignore_index=True)
    #return the panda DataFrame
    return(k)
```

Listing 1: File Function

```
def participant(file_path):
    #read the xml file
    infile = open(file_path,"r", encoding="utf8")
    contents = infile.read()
    soup = BeautifulSoup(contents,'xml')
    # branch in the file where you can find information on the participant
    p = soup.find_all('participant')
    # branch where the date of recording is mentioned (to calculate age
    later)
    f = soup.find_all('header')
    for j in f:
        dater = j.find('date')
    # create a dictionary and a panda DataFrame to stock data later
    dico ={'id':'', 'participant_role': '', "participant_age":''}
    participants = pd.DataFrame(columns = dico.keys())
    #retrieve the data
    for i in p:
        id1 = i['id']
        role = i.find('role')
        if role != None:
```

```
            role = i.find('role').contents
        age = i.find('age')
        # reformulate the age of the participant if the age is mentioned
        if age != None:
            for x in age:
                m= re.match(r'^P(\S+)Y(\S+)M(\S+)D(\S+)', x)
                age = m.group(1)+'/'+ m.group(2)+ '/'+ m.group(3)
        # in some data the age was not mentioned but we calculated it base
    on birthday information
        else:
            birthday = i.find('birthday')
            if birthday != None:
                for x in birthday:
                    m= re.match(r'^(\S+)-(\S+)-(\S+)', x)
                    bday = date(int(m.group(1)),int(m.group(2)), int(m.
    group(3)))
                for y in dater:
                    m = re.match(r'^(\S+)-(\S+)-(\S+)', y)
                    fday = date(int(m.group(1)),int(m.group(2)),int(m.group
    (3)))
                nyears, remainder = divmod((fday-bday).days, 365)
                nmonths, ndays = divmod(remainder, 30)
                age = "{}/{}/{}".format(nyears,nmonths,ndays)
        #dictionary with the implemented data
        dico= {'id': id1, "participant_role": role ,"participant_age": age}
        #add data to the DataFrame thank to a dictionary
        participants = participants.append(dico, ignore_index=True)
    #return the panda DataFrame
    return participants
```

Listing 2: Participant Function

```
def occurrence(file_path):
    tagger = treetaggerwrapper.TreeTagger(TAGLANG='fr')
    #read the xml file
    infile = open(file_path,"r", encoding="utf8")
    contents = infile.read()
    soup = BeautifulSoup(contents,'xml')
    # branch in the file where you can find information on the sentence
    u = soup.find_all('u')
    # create the dictionaries and a panda DataFrames to stock data later
    dico0 = {'sentence_id':'', 'speaker':'', 'sentence':''}
    dico1 = {'sentence_id': '', 'speaker':'', 'sentence':'', 'word':'', '
```

```
    ipa-actual':'', 'ipa-model':'',
12              'POS':'', 'lemma':'', "morphology":""}
13      dic0 = pd.DataFrame(columns = dico0.keys())
14      dic1 = pd.DataFrame(columns = dico1.keys())
15      # retrieve the data
16      for i in u:
17          # don't consider utterance where the speaker is unknown
18          if i.get('speaker') == None:
19              continue
20          # if we know the speaker we retrieve data
21          sp = i['speaker']
22          id1 = i['id']
23          ort = i.find('orthography')
24          sentence = list(itertools.chain(*(ort.find_all('w'))))
25          # ipa extraction
26          ipa = i.find(form = "actual")
27          ipa_a = list(itertools.chain(*(ipa.find_all('w'))))
28          ipa = i.find(form = "model")
29          ipa_m = list(itertools.chain(*(ipa.find_all('w'))))
30          # remove some character from sentence for more readability
31          chars = ['<', '>','0']
32          s = []
33          for j in sentence:
34              s.append(j.translate(str.maketrans({ord(x): '' for x in chars})
    ))
35          # fill the first dictionary with data regarding the utterance
36          dic = {'sentence_id':id1, 'speaker':sp, 'sentence':' '.join(s)}
37          dic0 = dic0.append(dic, ignore_index=True)
38
39          tags = tagger.tag_text(s, tagonly = True)
40          # morphology retrieval
41          morpho = i.find(tierName = "Morphology")
42          if morpho != None:
43              mor_w = list(itertools.chain(*(morpho.find_all('w'))))
44          else:
45              mor_w = [None]
46          # align data to create second dictionary
47          for (i,j,k,x,y) in itertools.zip_longest(sentence,ipa_a,ipa_m,tags,
    mor_w):
48              sentence0 = ' '.join(s)
49              # word in sentence
50              word = i
```

```
            if word != None:
                word = i.translate(str.maketrans({ord(x): '' for x in chars
    }))
            #POS and Lemma with TreeTagger
            t = []
            if x != None:
                m = re.match(r'^(\S+)\t(\S+)\t(\S+)', x)
                #word = m.group(1)
                if word == 'xxx' or word== 'yyy' or m == None:
                    pos = None
                    lemma = None
                    t.append((word,pos,lemma))
                else:
                    pos = m.group(2)
                    lemma = m.group(3)
                    t.append((word,pos,lemma))
                for l in t:
                    pos = l[1]
                    lemma = l[2]
                    # implement the second dataFrame with data
                    dic = {'sentence_id':id1, 'speaker':sp, 'sentence':
    sentence0, 'word':word,
                           'ipa-actual':j, 'ipa-model':k, 'POS':pos, 'lemma
    ':lemma, 'morphology':y}
                    dic1 = dic1.append(dic, ignore_index=True)
    #return the 2 dataFrame
    return(dic0, dic1)
```

Listing 3: Utterance and Occurrence Function

```
def createdb(h,u,p,db):
    #connect to the database
    mydb = mysql.connector.connect(host=h,
                                   user=u,
                                   password=p)
    mycursor = mydb.cursor()
    # to create the database named as the variable db
    mycursor.execute("CREATE DATABASE %s"%db)
```

Listing 4: create a database with mysql

```
def createtable(h,u,p,db):
    #connect to the database
    mydb = mysql.connector.connect(host=h,
```

```
                                    user=u,
                                    password=p,
                                   database = db)
    mycursor = mydb.cursor()

    # create the table 'collection'
    mycursor.execute("CREATE TABLE collection (id INT AUTO_INCREMENT
    PRIMARY KEY, name VARCHAR(255))")
    # create the table 'file'
    mycursor.execute("CREATE TABLE file (id INT AUTO_INCREMENT PRIMARY KEY,
     name VARCHAR(255), content LONGBLOB, collection_id INT ,FOREIGN Key (
    collection_id) REFERENCES collection(id))")
    # create the table 'participant'
    mycursor.execute("CREATE TABLE participant (id INT AUTO_INCREMENT
    PRIMARY KEY, role VARCHAR(255), age VARCHAR(255) ,file_id INT ,FOREIGN
    key (file_id) REFERENCES file(id) )")
    # create the table 'utterance'
    mycursor.execute("CREATE TABLE utterance (id INT AUTO_INCREMENT PRIMARY
     KEY, content BLOB, speaker INT, sentence_id VARCHAR(225), file_id INT,
    FOREIGN key (file_id) REFERENCES file(id),FOREIGN key (speaker)
    REFERENCES participant(id) )")
    # create the table 'occurrence'
    mycursor.execute("CREATE TABLE occurrence (id INT AUTO_INCREMENT
    PRIMARY KEY, word VARCHAR(255), ipa_model VARCHAR(255), ipa_actual
    VARCHAR(255), POS VARCHAR(255), lemma VARCHAR(255), morphology VARCHAR
    (255),participant_id INT, utterance_id INT , FOREIGN key (
    participant_id) REFERENCES participant(id), FOREIGN key (utterance_id)
    REFERENCES utterance(id))")
```

Listing 5: create the table in the database

```
def insertcollection(h,u,p,db):
    mydb = mysql.connector.connect(host=h,
                                    user=u,
                                    password=p,
                                   database = db)
    mycursor = mydb.cursor()

    sql = "INSERT INTO collection (id, name) VALUES (%s,%s)"
    val = [(0,'CHILDES-KernFrench'), (0,'CHILDES-Henkeler') , (0,'CHILDES-
    Lyon'), (0,'CHILDES-StanfordFrench'), (0,'COLAJE'), (0,'JEMS')]
    mycursor.executemany(sql,val)
    mydb.commit()
```

```
12
13      print(mycursor.rowcount, "record inserted.")
```

Listing 6: Insert Collection Name Into The Table

```
1  def insertdata(collection, h, u, p, db):
2      # The parameter collection of the function refers to the keys of our
       json dictionary
3      # The first part of the code retrieve the path to the file of the
       collection and put them in a list
4      with open('new_file.json', 'r') as f:
5          f = json.load(f)
6          f_p = []
7          for i in f[collection]:
8              for name, files in i.items():
9                  f_p.append(files[2])
10     #list with path
11     f_path = list(itertools.chain(*f_p))
12
13     # This part connect to the sql database that we have created before
14     # h, u, p, db correspond to the value you have chossen to connect to
       the database
15     mydb = mysql.connector.connect(host=h,
16                                    user=u,
17                                    password=p,
18                                    database=db
19                                   )
20     mycursor = mydb.cursor()
21     # this part apply the 3 functions created before on a file and store
       the result
22     for i in f_path:
23         f = file_(i)
24         p = participant(i)
25         o = occurrence(i)
26         u = o[0]
27         o0 = o[1]
28         # this part insert data into the table file
29         for j in range(len(f)) :
30             file_name = f.loc[j, "file_name"]
31             file_content = f.loc[j,"content"]
32             mycursor.execute("SELECT id FROM collection WHERE name ='COLAJE
       '")
33             myresult = mycursor.fetchall()
```

```
34            for x in myresult:
35                k = x[0]
36                sql = 'INSERT INTO file (id, name, content, collection_id)
    VALUES (%s, %s , %s , %s)'
37                fi= (0, file_name,file_content, k)
38                mycursor.execute(sql,fi)
39                mydb.commit()
40                # store the primary key id of the file, it will be later
    used into other table as a foreign key
41                fileid = mycursor.lastrowid
42
43        # insert the participant into the table and link them to the right
    file
44        participantf = []
45        for k in range(len(p)):
46            p_id = p.loc[k,'id']
47            p_role = ''.join(p.loc[k, 'participant_role'])
48            p_age = p.loc[k,'participant_age']
49            sql = 'INSERT INTO participant (id, role, age, file_id) VALUES
    (%s, %s , %s , %s)'
50            pi= (0, p_role,p_age, fileid)
51            mycursor.execute(sql,pi)
52            mydb.commit()
53            # store the primary key of the participant and the id of the
    participant in the file
54            participantid = mycursor.lastrowid
55            participantf.append((participantid,p_id))
56
57        # insert utterance into database row by row
58        # the table utterance have the columns: id, content, speaker,
    sentence_id, file_id
59        # sentence_id correspond to the utterence id in the file it will be
     usefull later to insert the occurrence
60        for l in range(len(u)):
61            utt_id = u.loc[l,'sentence_id']
62            utt_speaker = u.loc[l,'speaker']
63            utt_content = u.loc[l,'sentence']
64            # find the participant id of the person speaking
65            for j in participantf:
66                if utt_speaker == j[1]:
67                    speaker= j[0]
68            sql = "INSERT INTO utterance(id, content, speaker, sentence_id,
```

```
 file_id) VALUES (%s, %s , %s , %s, %s)"
            ui = (0, utt_content, speaker , utt_id, fileid)
            mycursor.execute(sql,ui)
            mydb.commit()

        # insert data into occurrence row by row
        # The table occurrence have the colums : id,word,POS,Lemma,
ipa_model, ipa_actual, morphology , participant_id, utterance_id
        for m in range(len(o0)):
            #utterance information to find the foreign key later
            utt_id = o0.loc[m,'sentence_id']
            #occurence0 info to store into the database
            word= o0.loc[m,'word']
            pos= o0.loc[m,'POS']
            lemma = o0.loc[m, 'lemma']
            ipa_a = o0.loc[m, 'ipa-actual']
            if ipa_a != None:
                ipa_a = str(ipa_a)
            ipa_m = o0.loc[m, 'ipa-model']
            if ipa_m != None:
                ipa_m = str(ipa_m)
            morpho = o0.loc[m, 'morphology']
            if morpho != None:
                morpho = str(morpho)
            # find the utterance_id and participant_id of the occurence
            mycursor.execute('set GLOBAL max_allowed_packet=67108864')
            mycursor.execute("SELECT * from utterance where sentence_id = %
s and file_id = %s limit 1",(utt_id,fileid))
            myresult = mycursor.fetchall()
            for x in myresult:
                utterence_id = x[0]
                participant_id = x[2]

            sql = "INSERT INTO occurrence (id,word,POS,Lemma,ipa_model,
ipa_actual, morphology , participant_id, utterance_id) VALUES (%s, %s ,
 %s , %s, %s,%s, %s, %s, %s)"
            o0i = (0,word,pos,lemma,ipa_m,ipa_a,morpho,participant_id ,
utterence_id)
            mycursor.execute(sql,o0i)
            mydb.commit()
```

Listing 7: Insert Data Into The Database Under The CoLaJE Collection

Bibliography

References

[1] Lex.e.m.
URL https://projetlexem.wixsite.com/website

[2] What is a relational database (rdbms)?
URL https://www.oracle.com/database/what-is-a-relational-database/

[3] Childes.
URL https://childes.talkbank.org/

[4] Ortolang.
URL https://www.ortolang.fr/market/corpora

[5] H. Henkeler, Aspects of the evolution of the early lexicon in the interactions mother-child: Case study of two dizygotic twin children between 15 and 26 months. university of rouen. (2005).

[6] B. Davis, S. Kern, A. Vilain, C. Lalevée, Des babils à babel: les premiers pas de la parole., Revue Française de Linguistique Appliquée 13 (2) (2008) 81–91.

[7] K. Demuth, A. Tremblay, Prosodically-conditioned variability in children's production of french determiners., Journal of Child Language 35 (2008) 99–127.

[8] M. M. Vihman, C. A. Ferguson, M. Elbert, Phonological development from babbling to speech: Common tendencies and individual differences., Applied Psycholinguistics 7 (1) (1986) 3–40. doi:https://doi.org/10.1017/S0142716400007165.

[9] Colaje.
URL http://colaje.scicog.fr/

[10] B. Gavin, What Is An XML File (And How Do I Open One)? (07 2018).
URL https://www.howtogeek.com/357092/what-is-an-xml-file-and-how-do-i-open-one/

[11] Beautiful soup documentation.
URL https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[12] Treetagger- a part-of-speech tagger for many languages.
URL https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

[13] Treetagger python wrapper's documentation.
URL https://treetaggerwrapper.readthedocs.io/en/latest/

[14] re — Regular expression operations — Python 3.9.5 documentation.
URL https://docs.python.org/3/library/re.html

[15] Introduction to SQL (Structure Query Language) — Studytonight (2021).
URL https://www.studytonight.com/dbms/introduction-to-sql.php

[16] Sql syntax.
URL https://www.w3schools.com/sql/sql_syntax.asp

[17] Pandas Basics - Learn Python - Free Interactive Python Tutorial.
URL https://www.learnpython.org/en/Pandas_Basics