# NELL Project Report

**Tong Jiajun**

**Chen Ze**

**Dong Yu**

## I. Introduction

NELL (Never-Ending Language Learner) is a computer system that learns over time to read the web. We are given a dataset of NELL and are required to design and implement a database, supporting four given queries and one query designed by ourselves.

This report contains problem design, process of our experiment and some conclusion.

## II. Problem design

### i. Schema and table design

The whole aspect of data (the first three columns) is shown as follows.

| A<br>Entity<br>TEXT | B<br>Relation<br>TEXT | C<br>Value<br>TEXT |
|---|---|---|
| concept:actor:gemma_hayes | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Gemma%20Hayes |
| concept:island:calf_of_man | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Calf%20of%20Man |
| concept:geopoliticallocation:pampore | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Pampore |
| concept:awardtrophytournament:cyprus_national_foot... | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Cyprus%20national%20football%20te... |
| concept:buildingmaterial:bend | concept:mutualproxyfor | concept:physicalaction:oregon |
| concept:buildingmaterial:bend | concept:proxyfor | concept:book:new |
| concept:buildingmaterial:bend | concept:subpartof | concept:politicsissue:oregon |
| concept:chef:mary_johnson_bailey_lincoln | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Mary%20Johnson%20Bailey%20Lincoln |
| concept:automobilemodel:conversion_van | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Conversion%20van |
| concept:physiologicalcondition:oncothermia | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Oncothermia |
| concept:personafrica:andy_robin | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Andy%20Robin |
| concept:personcanada:yumi_sugimoto | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Yumi%20Sugimoto |
| concept:personsouthamerica:monogatari | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Monogatari |
| concept:personaustralia:liam_cahill | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Liam%20Cahill |
| concept:ethnicgroup:murle | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Murle%20people |
| concept:stadiumoreventvenue:foellinger_auditorium | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Foellinger%20Auditorium |
| concept:coach:fred_hucul | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Fred%20Hucul |
| concept:model:yasemin_ergene | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Yasemin%20Ergene |
| concept:beach:cow_bay | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Cow%20Bay,%20Nova%20Scotia |
| concept:plant:calodendrum | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Calodendrum |
| concept:personaustralia:stuart_massey | concept:haswikipediaurl | http://en.wikipedia.org/wiki/Stuart%20Massey |
| concept:biotechcompany:corvel | concept:haswikipediaurl | http://en.wikipedia.org/wiki/CorVel%20Corporation |
| concept:county:johannes_kepler_university | concept:latitudelongitude | 48.3379200000000,14.3207100000000 |

We need to extract useful information in these columns and convert the result in to table that can be used for our query. So for each query we

need to design a table that represents the relationship between the input and output of the query.

### ii. Dataset design

The data we get has three major columns, which are the columns we concerns. So the first step on operation the data is extracting the first three columns out, both reducing the data size and accelerating the speed to process it.

To match the tables we designed earlier, we establish three sets of data. The first set is name-entity pair, for the first query that given a name, return all entities that match the name. In this set, we traverse all the tuples and find all the entities. Then we extract the name out of the entity and save it as a key.

The second set is entity-category pair, for the second and third queries. In this case, we get all the tuples with the relation “generalizations” and save the entities and categories in pairs. Both column are saved as key because those two queries ask for different input.

The third set is entity-entity pair, for the fourth query that given an entity, return all entities that are co-occurred with this entity in one triple. In this case, we find all triples with relations containing two entities and save them in pairs.

After extracting the data, we realized that there are redundancies because the raw data has lines with the same tuple. So we search in the three set and remove the redundancy.

### iii. Query design

For the four queries required to be supported, we design them as follows.

Query1: given a name, return all the entities that matches the name.

We only need the first column when creating table for this query. The easiest way is to create a table with entities and corresponding name. But, we can see that entities are in the form like (concept:buildingmaterial:bend), which is not atomic. So we create a table with the mid part of the entity and name (the last part) to keep attributes atomic.

Query2: given a category, return top 50 entities belongs to this category.

We need the first and third column to be the attributes of the table. When creating the table we select the rows with relation value 'generalization' and remove the repeated parts. For the conciseness of the table we also removed the needless word 'concept' appearing in each attributes.

Query3: given an entity, return all categories it belongs to.

Actually, the table for query3 is as same as which for query2. They are requesting on the same relationship. And Query3 take entity as the input and category as the output but query2 takes the other way round.

Query4: given an entity, return all entities that occurs in the same triple with it.

The table for this query is very similar to the previous one. We need the first and third column to be the attributes again. But we don't simply put all the rows of the given data into the table, because some of them may not have an entity as the value of the third column. So when creating this table we only retain rows having entity as the value of 'Value' column.

And the query we design ourselves is that given a name, return all categories that it belongs to. For this query, we need to nature join two of the dataset we design and select rows in the join set.

## III. The experiment

### i. Insert data into database

The database we use is mysql, and the scripting language we use is python version 2.7. On our own laptops, the database is named as "mydb", and we log in as root user, so as to conveniently operate the database. With the library MySQLdb, we establish connection with the database by the following code.

```
db = MySQLdb.connect("localhost","root","123456","mydb")
cursor=db.cursor()
```

Then, we create tables we designed. An example is here.

```
sql = """CREATE TABLE ENTITY_CATEGORY (
          ENTITY  VARCHAR(350) NOT NULL,
          CATEGORY  VARCHAR(350) NOT NULL,
          INDEX _ENTITY (ENTITY) USING HASH,
          INDEX _CATEGORY (CATEGORY) USING HASH)"""
try:
    cursor.execute(sql)
except:
    print("Table ENTITY_CATEGORY already exists.")
```

As we detected earlier, the longest entity is over 300 letters' long, we define the max length 350. Then we make index for the keys.

The next step is inserting data into the database, which takes the longest time. We first try to import data file with form '.csv' strictly into the database, but it seems to take too long with the speed of about 10 records per second. Then we try another approach, that inserting each record into the database with the help of script. However, it also cost too much time, with the speed about 1000 records per minute.

After searching on the internet, we found out that the operation "Insert Into" takes much longer time than the insert operation itself, and that an insert carry can contain multiple value tuples. Then we decide to write a sql query for inserting in a file, containing multiple records, thus reducing the times to do the "insert" linking. Then we meet another problem that making the query may takes too long, and the link with database may break because of long time waiting. What's more, sql does not support such long query. After times of trying, we finally choose 40000 as the number of records to insert in a query.

```
for line in DATA:
    if cnt==0:
        midfile = open("midfile",'w')
        midfile.write("INSERT INTO ENTITY_CATEGORY(ENTITY,CATEGORY) VALUES")
    datapair = line[:-1].split(',')
    midfile.write("('"+datapair[0]+'\',\''+datapair[1]+"'),\n")
    cnt=cnt+1
    if cnt==40000:
        midfile.close()
        f=open("midfile",'r')
        ct=f.readlines()
        sql=''.join(ct)[:-2]+';'
        cursor.execute(sql)
        db.commit()
        midfile.close()
        cnt=0
midfile.close()
f=open("midfile",'r')
ct=f.readlines()
sql=''.join(ct)[:-2]+';'
cursor.execute(sql)
db.commit()
midfile.close()
print("entity category done")
```

This method greatly improve the speed of inserting data, to about 5 million records per minute.

### ii. UI part

In the first version of our NELL project, we make use of Tkinter in python to design our user interface, since our data processing is done in python as well. It seems that the ui program is simple and easy to use. Users can just run the program and the ui just occur. But it's obvious that the there are many shortcomings. First, users should get the ui program before they can visit the database. Second, any changes in our database can make the ui program not work. So take users experience into consideration, we decide to design our user interface in php/html.

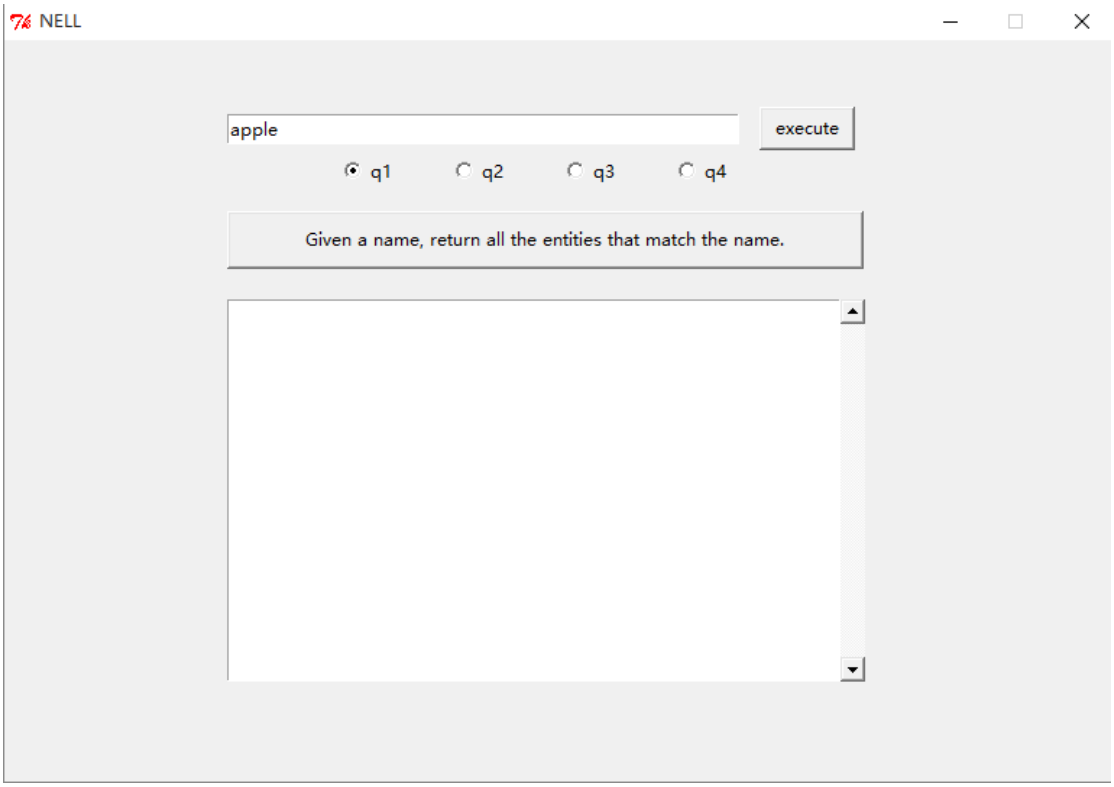
The interface is simple. When a user visits our website, there is only a title, an input box, a sets of query choices and a button to execute the query. The presentation of result is also very simple. You will see a sentence which tell us the number of records and the time taken to execution. Then the result of the query is contained in a table which consists of one or two column.

When we finished our ui design, we found out that there are many additional advantages of using php/html for ui design. First, users can visit the database whenever and wherever unless we close the server. Second, the visitation via website makes users easier to get

Second, visitation through website makes it easier for users to get their historical query records, as long as their browsers record cookies. Third, any changes we make to the database will never lead to failure visitation unless we break it down.

Although we finally choose php, but we will submit both python version and php version.

Here is the screenshot of python version ui:




And the screenshot of php version ui:

### iii. Query part

There are Five queries to design.

The first one is that given a name, return all the entities that match the name. For this query, a select sql query is to be execute.

```
case 1:
    $qry = "SELECT * FROM NAME_ENTITY WHERE NAME='" . $kw . "'";
    $result = mysqli_query($con,$qry);
    $t2= getMillisecond();
    $time = $t2-$t1;
```

The second one is that given a category, return top 50 entities belongs to this category. For this query, we select all entities belonging to this category and print the top 50.

```
case 2:
    $qry = "SELECT  * FROM ENTITY_CATEGORY WHERE CATEGORY='" . $kw . "' LIMIT 50";
    $result = mysqli_query($con,$qry);
    $t2= getMillisecond();
    $time = $t2-$t1;
```

The third one is given an entity, return all categories it belongs to.

```
case 3:
    $qry = "SELECT * FROM ENTITY_CATEGORY WHERE ENTITY='" . $kw . "'";
    $result = mysqli_query($con,$qry);
    $t2= getMillisecond();
    $time = $t2-$t1;
```

The fourth one is given an entity, return all entities that occurs in the same triple with it.

```
case 4:
    $qry = "SELECT * FROM ENTITY_ENTITY WHERE ENTITY1='" . $kw . "'";
    $result = mysqli_query($con,$qry);
    $t2= getMillisecond();
    $time = $t2-$t1;
```

And the query we designed ourselves is that given a name, return all the categories it belongs to.

```
case 5:
    $qry = "SELECT * FROM (SELECT * FROM NAME_ENTITY WHERE NAME='" . $kw . "') A NATURAL JOIN ENTITY_CATEGORY";
    $result = mysqli_query($con,$qry);
    $t2= getMillisecond();
    $time = $t2-$t1;
```

### iv. Some of the searching results

# N E L L

apple Query

Name_to_Entity Category_to_Entity Entity_to_Category Entity_to_Entity Name_to_Category

Totally 4 results! Execution time:1ms.

| entity |
| --- |
| biotechcompany:apple |
| plant:apple |
| company:apple |
| bank:apple |

# N E L L

person Query

Name_to_Entity Category_to_Entity Entity_to_Category Entity_to_Entity Name_to_Category

Totally 50 results! Execution time:255ms.

| entity |
| --- |
| person:rose |
| person:joe |
| person:regina |
| visualizablescene:robinson |
| person:massa |
| person:charles |
| visualizablescene:johnston |
| person:marco |
| person:button |
| person:thai |
| person:robert_w__ingram |
| person:st__hyginus |
| person:julius_nyerere |
| person:john_redd |
| person:ming_lee |
| person:david_kopras |

---

# N E L L

rose Query

Name_to_Entity Category_to_Entity Entity_to_Category Entity_to_Entity Name_to_Category

Totally 5 results! Execution time:270ms.

| category |
| --- |
| food |
| plant |
| person |
| visualizablething |
| beverage |

## IV. Conclusion

In this project we established a database using the given data. And we designed several usable queries for the database. We have definitely benefited a lot from this project. Through this project we learned about data processing with Python , importing data with Mysql and design UI with php. We have got a clearer conception about how database works after finishing all these works.

## V. Some statements

The database we use in our experiment is MySQL server, version 5.7.

The tool we use to import data is MySQLdy library in python 2.7.

The language we use to build the UI is PHP.

The local host on our laptop is: 59.78.48.65/nell.php, but we may not start it at any time.