BIOMEDQUERY.JL

Isabel Restrepo, PhD | PHP 2561, Brown University | April 25, 2017

Slides: https://gitpitch.com/bcbi/julia_tutorials/master?p=biomedquery

SET UP BEFORE CLASS

- Install Docker
- Make sure Docker Daemon is running
- Download Docker Image bcbi/julia_edu

```
docker pull bcbi/julia_edu:latest
```

- Is everyone done with these steps?

AWS?

http://34.207.254.102:8888/

hcwang | kjeong | kjline | pc16 | vdantu

http://54.173.109.173:8888/

amtran | bbqu | bmle | jsleung | mquinn | nchou

FINISH SETTING UP (IN CLASS)

- Make a directory where you will save your Jupyter notebooks. E.g.,

```
cd ~
mkdir php_2561
cd php_2561
mkdir tutorial_notebooks
```

- Run docker image while sharing your notebook directory

```
docker run -it --name bcbi_julia_edu -p 8888:8888  -v ~/php_2561/tutorial_notebooks
```

...FINISH SETTING UP (IN CLASS)

- Run Jupyter: Inside the container,

```
./run_jupyter.sh
```

- To open Jupyter visit http:/localhost:8888

WHAT IS BIOMEDQUERY.JL?

BioMedQuery.jl is a Julia package with utilities to interact with BioMedical Databases and APIs. Supported databases/APIS include:

- Entrez Programming Utilities (E-utilities)
- Unified Medical Language System (UMLS)
- Clinical Trials (dot) Gov
- Medical Text Indexer (MTI)

WHERE CAN I FIND DOCUMENTATION?

Documentation lives here:
http://bcbi.github.io/BioMedQuery.jl/stable/

WHAT IF THE FUNCTIONALITY I'M LOOKING FOR DOESN'T EXIST?

Submit a pull request here: https://github.com/bcbi/BioMedQuery.jl/pulls)

ENTREZ UTILITIES (EUTILS)

BiomedQuery.Entrez provides an interface to some of the functionality in the Entrez Utility API.

The following E-utils functions have been implemented:

- ESearch
- EFetch
- ELink
- ESummary

FUNCTIONS AVAILABLE TO HANDLE AND STORE NCBI RESPONSES

- EParse - Convert XML response to Julia Dict
- Saving NCBI Responses to XML
- Saving EFetch to a SQLite database
- Saving EFetch to a MySQL database
- Saving EFetch to a publication file (bibtex or endnote)

FUNCTIONS AVAILABLE TO QUERY THE DATABASE

- All PMIDs
- All MESH descriptors for an article

BEFORE WE START

(AWS users ... skip)

- Attach another interactive shell to your docker container

```
docker exec -it bcbi_julia_edu /bin/bash
```

- Create your .juliarc.jl file

```
cd ~
touch .juliarc.jl
```

(AWS users ... skip)

- Write your environment variables

```
emacs .juliarc.jl
```

```
Type the following ENV variables

ENV["NCBI_EMAIL"]="first_last@brown.edu"
ENV["UMLS_USER"]="user"
ENV["UMLS_PSSWD"]="password"

To save: ctrl-x ctrl-s
To quit: ctrl-x ctrl-c
```

- Start mysql service

```
sudo /etc/init.d/mysql start
```

LET'S START WITH ENTREZ

Create a Julia notebook called entrez

IMPORT THE MODULE AND ENVIRONMENT VARIABLES

- Regular users:

```
using BioMedQuery.Entrez
email = ENV["NCBI_EMAIL"];
umls_user = ENV["UMLS_USER"];
umls_psswd = ENV["UMLS_PSSWD"];
```

- AWS users:

```
using BioMedQuery.Entrez
email = "your email";
umls_user = "your umls user";
umls_psswd = IJulia.readprompt("UMLS password", password=true);
```

ESEARCH

Request a list of UIDS matching a query from an input dictionary specifying all required parameters specified in the Entrez documentation NCBI Entrez:Esearch.

EXAMPLE

Request 10 pmids for papers matching the query:

(asthma[MeSH Terms]) AND ("2001/01/29"[Date - Publication] : "2010"[Date - Publication])

```
search_term = """(asthma[MeSH Terms]) AND ("2001/01/29"[Date - Publication] : "20
search_dic = Dict("db"=>"pubmed", "term" => search_term,
"retstart" => 0, "retmax"=>10,
"email" => email)
esearch_response = esearch(search_dic)
```

SAVE THE RESPONSE TO FILE

```
using XMLconvert
xmlASCII2file(esearch_response, "./esearch.xml");
```

CONVERT TO A JULIA (MULTI) DICTIONARY

```
esearch_dict = eparse(esearch_response)
println("Type of esearch_dict: ", typeof(esearch_dict))
show_key_structure(esearch_dict)
```

FLATTEN INTO DICTIONARY FOR EASY ACCESS

```
flat_easearch_dict = flatten(esearch_dict)
display(flat_easearch_dict)
```

GET ALL PMIDS RETURNED BY ESEARCH

```
ids = Array{Int64,1}(flat_easearch_dict["IdList-Id" ])
```

EFETCH

```
# define the fetch dictionary
fetch_dic = Dict("db"=>"pubmed","tool" =>"BioJulia",
"email" => "maria_restrepo@brown.edu", "retmode" => "xml", "rettype"=>"null")

# fetch
efetch_response = efetch(fetch_dic, ids)
```

CONVERT TO XML RESPOSE TO (MULTI) DICTIONARY

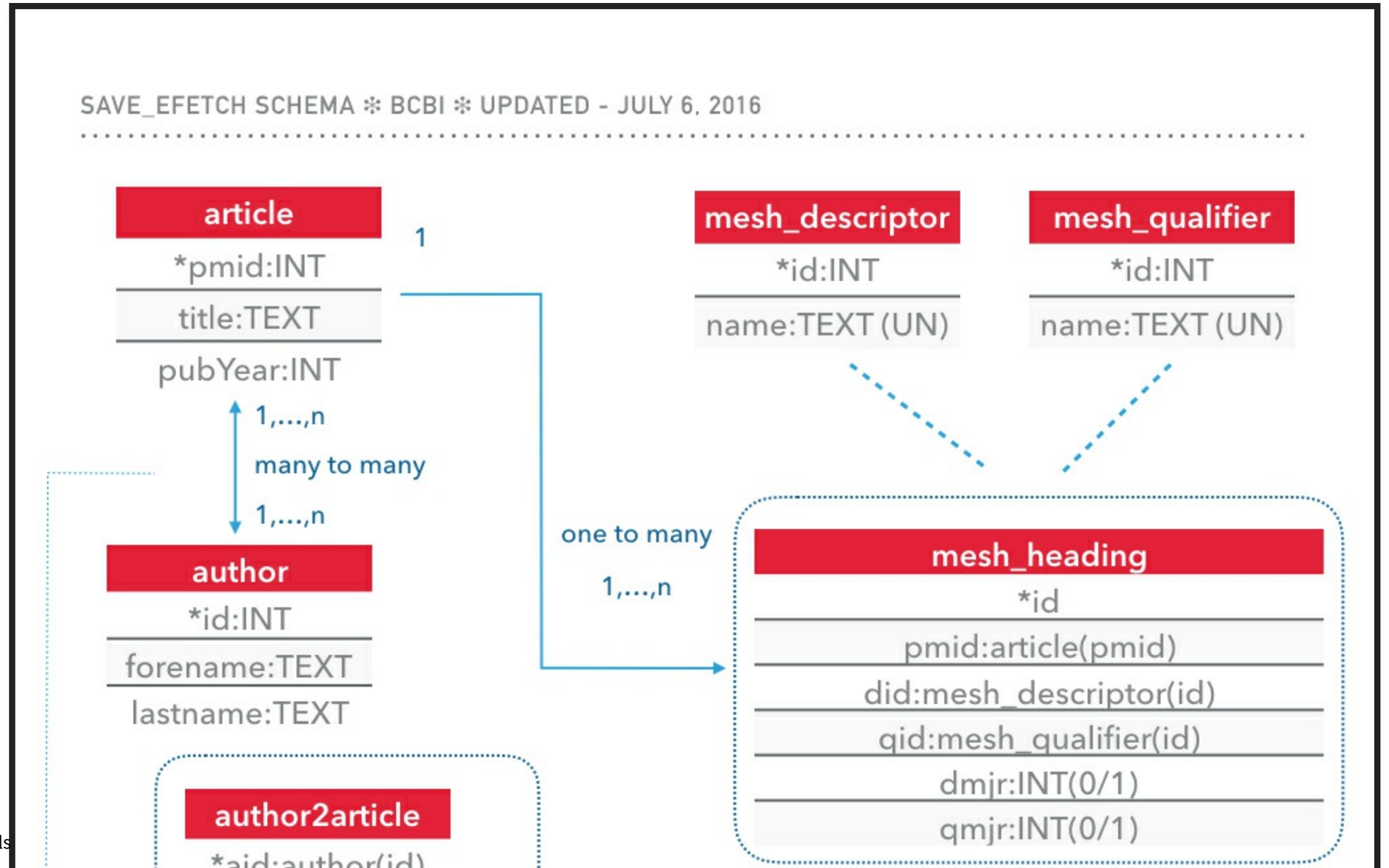
```
efetch_dict = eparse(efetch_response)
show_key_structure(efetch_dict)
```

SAVE TO MYSQL

```
db_config = Dict(:host=>"127.0.0.1",
                 :dbname=>"biomed_query_test",
                 :username=>"root",
                 :pswd=>"bcbi123",
                 :overwrite=>true)

db = save_efetch_mysql(efetch_dict, db_config)
```

MYSQL SCHEMA



EXPLORE THE MYSQL RESULTS DATABASE

```
using MySQL
tables = mysql_execute(db, "show tables;")
display(tables)
articles = mysql_execute(db, "select * from article limit 10")
display(articles)
authors = mysql_execute(db, "select * from author limit 10")
display(authors)
```

SAVE AS PUBLICATIONS

```
citation_config = Dict(:type => "bibtex", :output_file => "citations_test.bib", :overwrite=
    save_article_citations(efetch_dict, citation_config);
```

BIOMEDQUERY.PROCESSES

The library comes with a series a "pre-assembled" workflows. For instance, we often need to call esearc, efetch and save to database as a pipeline.

```
using BioMedQuery.Processes
```

ESEARCH, EFETCH, MYSQL_SAVE IN ONE LINE OF CODE

```
db = pubmed_search_and_save(email, search_term, 10,
    save_efetch_mysql, db_config);
```

ESEARCH, EFETCH, SAVE CITATIONS IN ONE LINE OF CODE

```
pubmed_search_and_save(email, search_term, 10,
    save_article_citations, citation_config);
```

LET'S START WITH UMLS

Create a Julia notebook called umls

BIOMEDQUERY.UMLS

Utilities to search the Unified Medical Language System (UMLS). This is a Julia interface to their REST API.

Searching the UMLS requires approved credentials. You can sign up here

As of today, the following utilities are available:

- verify credentials / issue umls tickets
- search_umls
- get the best maching cui from a query
- get the semantic type

SET UP

```
using BioMedQuery.UMLS
user = ENV["UMLS_USER"];
psswd = ENV["UMLS_PSSWD"];
credentials = Credentials(user, psswd)
query = Dict("string"=>"asthma", "searchType"=>"exact")
```

GET A TICKET AND SUBMIT QUERY

```
tgt = get_tgt(credentials)
all_results = search_umls(tgt, query)
```

GET BEST MATCHING CUI AND IT'S SEMANTIC TYPE

```
cui = best_match_cui(all_results)
display(cui)
sm = get_semantic_type(tgt, cui)
display(sm)
```

PROCESSES AVAILABLE FOR UMLS

- Get all UMLS semantic types for all MeSH stored in a database corresponding to results from an Entrez query

```
using BioMedQuery.Processes
using MySQL

db_host = "127.0.0.1"
mysql_usr = "root"
mysql_pswd = "bcbi123"
dbname = "biomed_query_test"

db = mysql_connect(db_host, mysql_usr, mysql_pswd, dbname)

map_mesh_to_umls_async!(db, credentials)
```

```
tables = mysql_execute(db, "show tables;")
display(tables)
```

- Filter by semantic type: For all articles in the 'results' database, filter all MeSH associated with a specific semantic type

```
labels2ind, occur = umls_semantic_occurrences(db, "Disease or Syndrome")

println("------------------------------------------------------------")
println("Output Descritor to Index Dictionary")
display(labels2ind)
println("------------------------------------------------------------")

println("------------------------------------------------------------")
println("Output Data Matrix")
display(full(occur))
println("------------------------------------------------------------")
```

PLOT CONDITIONAL PROBABILITIES AS A HISTOGRAM

```
collect(keys(labels2ind))
```

```
using PlotlyJS

trace1 = bar(;x=collect(keys(labels2ind)),
             y=sum(occur, 2)[:]./10,
             marker=attr(color="rgba(50, 171, 96, 0.7)",
             line=attr(color="rgba(50, 171, 96, 1.0)", width=2)))

data = [trace1]
layout = Layout(;margin_b = 100,
                 margin_r =100)

plot(data, layout)
```