ReadMe

* ***Introduction:*** ReadMe explains how we will extract OMOP Common Data Model (CDM) from Anthem LPR (json files).
* ***FHIR TO OMOP PROJECT:*** The aim of project is to use Anthem’s Fast Healthcare Interoperability Resources (FHIR) data, derived from the delivery of health care in routine clinical settings, to build a data infrastructure for:
  + Conducting patient-centered analysis
  + Cohort Building
* ***Scope:***
  + Extract tables for the existing resource type which are converted to FHIR i.e. Person, Condition, Visit, Observation, Medication Dispense and Practitioner. We are using EMR cluster for execution of all the files. ***Note:*** OMOP tables schema information can be retrieved partially form LPR files and remaining is to be retrieved from Athena files.
* ***Installations:*** 
  + SecureCRT 8.5 (22.174.142.245)
  + JetBrains PyCharm 2019.2.3
* ***Files used:***
* create\_cdmraw\_test\_tables.ddl
* fhir2omop\_new.py
* pysparkwrapper\_cdl.py
* pysparkwrapper\_cda.py
* runOMOPextraction\_cdl.sh
* runOMOPextraction\_cda.sh
* ***Execution Steps:***
* **Creating the schema**
  + - Using create\_cdmraw\_test\_tables.ddl, schema is created for both CDL and CDA files.
    - Using hive on the cluster for creating following test tables in the given S3 location: Person, Observation, Conditions, Medication Dispense, Procedures, Visit, Practitioner.
    - Command used: hive -f create\_cdmraw\_test\_table.ddl
  + **Running the code using shell script –** 
    - **Files – runOMOPextraction\_cdl.sh** or **runOMOPextraction\_cda.sh**
    - Pass Python code files (**pysparkwrapper\_cda.py** or **pysparkwrapper\_cdl.py**) in the shell script.
    - It has sparksubmit.zip file managing all Spark dependencies and use its capabilities.
  + **Mapping to target resource(OMOP) tables –** 
    - **Files – pysparkwrapper\_cda.py** or **pysparkwrapper\_cdl.py**
    - Here we are giving the cda and cdl lpr files as input and the ouput is a dataframe containing individual resource dataframes of following resources: Person, Observation, Conditions, Medication Dispense, Procedures, Visit, Practitioner.
    - Extracting LPR(s) from the database and converting into a dataframes containing rows of extracted information according to resource types using fhir2omop\_new.py file and further using User defined function(udf) which uses extractCDM() method.
    - get\_key() method uses explode function to flatten the columns in a dataframe into multiple columns. We are using get\_key() to explode the columns generated while flattening the rows in LPR and then further using it again to further flatten the columns/field\_names in every resource type.
    - Making a list of all the resources as attributes. Flatten all the attributes fields according to columns and get\_key() method.
    - Inserting and mapping the values into the target table made in the hive.
    - The schema of the test tables is made in the ddl file, create\_cdmraw\_test\_tables.ddl.
    - The python file is importing fhir2omop\_new.py to be used to extract predefined paths and to get required information as per resource type.
  + **Parsing Json – Using fhir2omop\_new.py file**
    - Pre-defined file paths are defined for such key: value data pairs to be extracted.
    - Specific key: value pairs from the Json file are required for a given resource type.
    - Json structure is differentiated based on whether the type is Dictionary, List, String or Float.
    - Function ‘get\_fields\_from\_dict’ helps in making a dictionary for the keys according to the field list. If field name, eg. ‘mcid’ is in the extracted fields from json, this function will populate the dictionary of that resource with ‘mcid’, else it will return null.
    - Function ‘extract\_recursive’ checks whether the value is dictionary, list, string, or float. It runs recursively until it gets a string value, which is then selected and stored.
    - The output is a list of dictionary (e.g. person, condition, visit etc.). These will constitute rawCDM tables’ column and rows

**Reading Athena Files:**

1. **Files**
   * create\_athena\_tables.ddl
   * readATHENA.py
   * readATHENA.sh
2. **Creating Schema**
   * Command– hive -f create\_athena\_tables.ddl
   * Uses create\_athena\_tables.ddl to create Athena tables (concept, concept\_ancestor, concept\_relationship). Queries are run on Hive and have the AWS S3 location wherein respective folders get created.
3. **Reading Athena files and converting into dataframe**
   * Python application code ‘readATHENA.py’ is submit to cluster using the ‘readATHENA.sh’ spark-submit script.
   * ‘readATHENA.py’ file uses PySpark to read the Athena csv files (CONCEPT.csv, CONCEPT\_ANCESTOR.csv, CONCEPT\_RELATIONSHIP.csv) stored in AWS S3 location and converts them into dataframe.