Scientific Publications Data Warehouse

Project 1: A Data cube on top of Delta lake (ETL)

Purpose

The purpose is to extract data about scientific publications from JSON data that describe, title, topic, authors, etc. about a huge number of papers and populate a data warehouse in order to issue analytics queries using SQL.

Usage

Upload this notebook or DBC archive on databricks platform

Task1. Extract

a. Fetching The 7z archive

Skip this Section if you already have performed the extraction process and jump to checkpoint for pulling data from split json files.

In [0]:
```
# Checking if archive is downloaded in memory.
try:
    dbutils.fs.ls("file:/databricks/driver/dblp.v13.7z")
    print("Archive in filesystem (file:/databricks/driver/dblp.v13.7z)")
except:
    # If archive is not in memory, Checking databricks store for cached version and pulling int
    try:
        dbutils.fs.ls("dbfs:/FileStore/data/dblp.v13.7z")
        print("Archive located in FileStore. Copying into local store..")
        dbutils.fs.cp("dbfs:/FileStore/data/dblp.v13.7z", "file:/databricks/driver/dblp.v13.7z"
        print("Completed")
    except:
        # If archive is not cached, downloading and storing in databricks store.
        print("7z archive not found. Fetching from URL...")
        !wget https://originalstatic.aminer.cn/misc/dblp.v13.7z
        print("7z archive Downloaded. Moving archive to FileStore..")
        dbutils.fs.mkdirs("dbfs:/FileStore/data")
        dbutils.fs.cp("file:/databricks/driver/dblp.v13.7z", "dbfs:/FileStore/data/dblp.v13.7z"
        print("Completed.")
```

In [0]:
```
# The returned array should have one object of FileInfo with size =2568255035

dbutils.fs.ls("file:/databricks/driver/dblp.v13.7z")
```

b. Extracting Archive into json chunks

b1. Extracting 7zip file into json.

In [0]:
```
!pip install py7zr -q
```

In [0]:
```
import py7zr

archive = py7zr.SevenZipFile('dblp.v13.7z', mode='r')
archive.extractall()
archive.close()
```

In [0]:
```
dbutils.fs.ls("file:/databricks/driver/dblpv13.json")
```

b2. Cleaning NumberInt(#) tags

The json data contains non-confirming tags, and so cannot be parsed as it is. We will read each line and substitute the tag. (This should take about 25 minutes)

In [0]:
```
import re

# Cleaning the `NumberInt` tag
fin = open(f"dblpv13.json")
fout = open(f"dblpv13_clean.json", "wt")
for line in fin:
    fout.write(re.sub(r"NumberInt\([\d]*\)", lambda x: "".join(re.findall(r"\d", x.group(0))),
fin.close()
fout.close()
```

b3. Partitioning Dataset into JSON files

Since the whopping 16 GB of json data cannot be loaded into memory directly, we need to partition the data into smaller chunks (300k objects per chunk) for processing.
We also parse data encoded as Decimal data with DecimalEncoder.

In [0]:
```
%mkdir data
```

In [0]:
```
import ijson
import json
import decimal

class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            return str(o)
        return super(DecimalEncoder, self).default(o)

data_dir = 'data/'
with open('dblpv13_clean.json', 'r') as f:
    counter, file_id = 0, 0
    file_buffer = []
    for obj_data in ijson.items(f, 'item'):
        file_buffer.append(obj_data)
        counter += 1
        if counter % 300000 == 0:
            print(f" Saving, data_PART_{file_id}.json in {data_dir}")
            f = open(f'{data_dir}data_PART_{file_id}.json', 'w')
            dump = json.dumps(file_buffer, cls=DecimalEncoder)
            f.write(dump)
            f.close()
            file_id += 1
            file_buffer = []
f = open(f'{data_dir}data_PART_{file_id}.json', 'w')
dump = json.dumps(file_buffer, cls=DecimalEncoder)
print(f" Saving, data_PART_{file_id}.json in {data_dir}")
f.write(dump)
f.close()
file_id += 1
file_buffer = []
```

b4. Moving files to dbfs FileStore from instance storage. Building Checkpoint.

In [0]:
```
# removing old json stored in filestore.
dbutils.fs.rm("dbfs:/FileStore/data/split_data/", recurse = True)
# Creating dir to store json in filestore..
dbutils.fs.mkdirs("dbfs:/FileStore/data/split_data")
# confirming dir is empty
dbutils.fs.ls("dbfs:/FileStore/data/split_data")
```

In [0]:
```
# Copying all json parts into filestore.
dbutils.fs.cp("file:/databricks/driver/data/", "dbfs:/FileStore/data/split_data", recurse = Tru
```

Task2. Transform

Goal: Read data from databricks Filestore into dataframes (Checkpoint after data load)

In [0]:

```
import uuid
from functools import reduce
import pyspark.sql.functions as F
from pyspark.sql.types import StructType, ArrayType, StringType, LongType, StructField, IntegerT
from typing import List
from pyspark.sql.functions import udf

# Here Path indicates input file path, and delta_dir points to file
path = "dbfs:/FileStore/data/split_data/"
delta_dir = "dbfs:/delta/tables/"

# There should be 18 files each with 300 k records. This would change if you change split value
file_count = len(dbutils.fs.ls(path))
assert file_count == 18, "Data not found. You may want to check the path or run the notebook fr
```

In [0]:

```
# Build map of spark dataframes by reading json partition chunk files
dataframes_map = map(lambda r: spark.read.option("inferSchema", True).json(r), [f"{path}data_PA
# reduce the dataframes into single dataframe by performing union over the mapped frames.
union = reduce(lambda df1, df2: df1.unionByName(df2, allowMissingColumns=True), dataframes_map)

# Reading first chunk for Testing
# union = spark.read.option("inferSchema", True).json(f"{path}data_PART_0.json")

# jsonSchema = StructType([
#               StructField("_id", StringType(), True),
#               StructField("abstract", StringType(), True),
#               StructField("authors", ArrayType(StructType([
#                       StructField("_id", StringType(), True),
#                       StructField("bio", StringType(), True),
#                       StructField("email", StringType(), True),
#                       StructField("gid", StringType(), True),
#                       StructField("name", StringType(), True),
#                       StructField("name_zh", StringType(), True),
#                       StructField("oid", StringType(), True),
#                       StructField("oid_zh", StringType(), True),
#                       StructField("orcid", StringType(), True),
#                       StructField("org", StringType(), True),
#                       StructField("org_zh", StringType(), True),
#                       StructField("orgid", StringType(), True),
#                       StructField("orgs", ArrayType(StringType(), True), True),
#                       StructField("orgs_zh", ArrayType(StringType(), True), True),
#                       StructField("sid", StringType(), True)
#               ]), True), True),
#               StructField("doi", StringType(), True),
#               StructField("fos", ArrayType(StringType(), True), True),
#               StructField("isbn", StringType(), True),
#               StructField("issn", StringType(), True),
#               StructField("issue", StringType(), True),
#               StructField("keywords", ArrayType(StringType(), True), True),
#               StructField("lang", StringType(), True),
#               StructField("n_citation", LongType(), True),
#               StructField("page_end", StringType(), True),
#               StructField("page_start", StringType(), True),
#               StructField("pdf", StringType(), True),
#               StructField("references", ArrayType(StringType(), True), True),
#               StructField("title", StringType(), True),
#               StructField("url", ArrayType(StringType(), True), True),
#               StructField("venue", StructType([
#                       StructField("_id", StringType(), True),
#                       StructField("issn", StringType(), True),
#                       StructField("name", StringType(), True),
#                       StructField("name_d", StringType(), True),
#                       StructField("name_s", StringType(), True),
#                       StructField("online_issn", StringType(), True),
#                       StructField("publisher", StringType(), True),
#                       StructField("raw", StringType(), True),
#                       StructField("raw_zh", StringType(), True),
#                       StructField("sid", StringType(), True),
#                       StructField("src", StringType(), True),
#                       StructField("t", StringType(), True),
#                       StructField("type", LongType(), True)
#               ]), True),
#               StructField("volume", StringType(), True),
#               StructField("year", LongType(), True)
#       ])


# union = spark.readStream.schema(jsonSchema).option("maxFilesPerTrigger", 1).json(path)

union = union.na.drop(subset=["authors"])
union = union.dropDuplicates(["_id"])
union = union.filter(union.lang == 'en')
union.printSchema()
```

```
root
 |-- _id: string (nullable = true)
 |-- abstract: string (nullable = true)
 |-- authors: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- _id: string (nullable = true)
 |    |    |-- bio: string (nullable = true)
 |    |    |-- email: string (nullable = true)
 |    |    |-- gid: string (nullable = true)
 |    |    |-- name: string (nullable = true)
 |    |    |-- name_zh: string (nullable = true)
 |    |    |-- oid: string (nullable = true)
 |    |    |-- oid_zh: string (nullable = true)
 |    |    |-- orcid: string (nullable = true)
 |    |    |-- org: string (nullable = true)
 |    |    |-- org_zh: string (nullable = true)
 |    |    |-- orgid: string (nullable = true)
 |    |    |-- orgs: array (nullable = true)
 |    |    |    |-- element: string (containsNull = true)
 |    |    |-- orgs_zh: array (nullable = true)
 |    |    |    |-- element: string (containsNull = true)
 |    |    |-- sid: string (nullable = true)
 |    |    |-- position: string (nullable = true)
 |    |    |-- avatar: string (nullable = true)
 |    |    |-- homepage: string (nullable = true)
 |-- doi: string (nullable = true)
 |-- fos: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- isbn: string (nullable = true)
 |-- issn: string (nullable = true)
 |-- issue: string (nullable = true)
 |-- keywords: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- lang: string (nullable = true)
 |-- n_citation: string (nullable = true)
 |-- page_end: string (nullable = true)
 |-- page_start: string (nullable = true)
 |-- pdf: string (nullable = true)
 |-- references: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- title: string (nullable = true)
 |-- url: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- venue: struct (nullable = true)
 |    |-- _id: string (nullable = true)
 |    |-- issn: string (nullable = true)
 |    |-- name: string (nullable = true)
 |    |-- name_d: string (nullable = true)
 |    |-- name_s: string (nullable = true)
 |    |-- online_issn: string (nullable = true)
 |    |-- publisher: string (nullable = true)
 |    |-- raw: string (nullable = true)
 |    |-- raw_zh: string (nullable = true)
 |    |-- sid: string (nullable = true)
 |    |-- src: string (nullable = true)
 |    |-- t: string (nullable = true)
 |    |-- type: long (nullable = true)
 |-- volume: string (nullable = true)
 |-- year: string (nullable = true)
```

Cleaning bad records (empty author lists, small titles)

In [0]:
```
# Deleting entries with small Titles (less than 3 words) and empty author list
size_ = udf(lambda s: len(s.split()), IntegerType())

union = union.na.drop(subset=["title", "authors"])
union = union.filter(size_(F.col("Title")) > 3)
```

Reading the dataframe by merging the previously created chunks.

Alternatively, we can process single chunk to see what outcome may look like

In [0]:
```
def save_delta_frame(frame, alias, clean = False):
    # pull required Fields
    delta_path=f"{delta_dir}{alias}"

    # Clean (delete dups, Fill NaN values?, ...)
    if clean:
        frame = frame.distinct()

    # Save delta Frame
    frame.write.format('delta').mode('overwrite').save(delta_path)

    # frame.writeStream.format('delta').option("checkpointLocation", f"/delta/{alias}/_checkpoi
    # pull appeneded delta file and return
    # frame = spark.read.format('delta').load(delta_path)
    return frame

def distinct_frame_from_cols(frame, columns):
    # get distinct records for col
    frame = frame.select(*columns).distinct()
    # frame = frame.select("*").withColumn("id", F.monotonically_increasing_id() + 1)
    frame = frame.select("*").withColumn("id", F.expr("uuid()"))
    # return the indexed Table
    return frame.select("id", *columns)

def map_rdd_to_id(rdd):
    def map_rdd2_id_(col):
        if col == "null" or col == "" or not col:
            return None
        try:
            return [rddTuple[0] for rddTuple in list(rdd.items()) if rddTuple[1] == col][0]
        except ValueError:
            return None
    return udf(map_rdd2_id_, LongType())

# UDF to get relevant publication's citation counts
def cite_count(countMapper):
    def cite_count_(col):
        if col == "null" or col == "" or not col:
            return "Unknown"
        return countMapper.get(col)
    return udf(cite_count_, StringType())
```

Language Table

- Counting number of distinct languages.
- Building new table.
- Saving Table to Delta lake

In [0]:
```
lang_frame = distinct_frame_from_cols(union, ['lang']).withColumnRenamed("lang", "Text")
save_delta_frame(lang_frame, "Language")
lang_rdd = lang_frame.rdd.collectAsMap()

union = union.select("*", map_rdd_to_id(lang_rdd)("lang").alias("Lang_ID")).drop("lang")
```

Publication Table

- Counting number of citations.
- Building new table for Title, abstract, volume, Number of citations, references and more.
- Saving Table to delta lake

In [0]:
```
# building a Citation counter dictionary
citation_frame = union.select(F.explode_outer("references").alias("reference_countmap"))
citation_frame = citation_frame.groupBy("reference_countmap").count()
citation_frame = citation_frame.rdd.map(lambda row: row.asDict(True))
citation_counts = citation_frame.collect()
citation_counter = {}
for citation_count in citation_counts:
    citation_counter[citation_count['reference_countmap']] = citation_count['count']
```

In [0]:
```
# Building Publication Frame
publication_frame = union.select("_id", "title", "volume", "issue", "abstract", "pdf", "isbn",
publication_frame = publication_frame.withColumn("issn", F.when(publication_frame.issn.rlike("[
# Removing extracted fields from the main schema
union = union.drop("title", "abstract", "pdf", "isbn", "issn", "doi", "url", "references", "pag
```

In [0]:
```
# Saving the table
publication_frame = save_delta_frame(publication_frame, "Publication", clean=True)
```

In [0]:
```
### Future Steps:
# 1. API lookup to fill in missing data in issn, isbn, pdf columns
# 2. Extract distinct from doc_type into Type frame and map ID for the same
```

FieldOfStudy table.

- To generalize desciplines, initialize CountMapper and decipline_mapper.
- Map relevant `Field of study` topic for each record
  - If the decipline is found in the generalized mapper, we use that item to map the Field of Study list.
  - Otherwise we use counts of occurances of each item from the list in the whole database, and pick the one with most frequent occurance as a suitable discipline.

We used Suggested Descipline mappings to build a relevant datastructure to replace the specific field to generalized descipline.

In [0]:

```
# Building countmap structure
countMapFos = union.select(F.explode("fos").alias("fos2"))
countMapFos = countMapFos.groupBy("fos2").count()
countMapperRdd = countMapFos.rdd.map(lambda row: row.asDict(True))

countMapperList = countMapperRdd.collect()

count_mapper = {}
for countMapperItem in countMapperList:
    count_mapper[countMapperItem['fos2']] = countMapperItem['count']

decipline_mapper = {
        # 1 Natural Sciences
        "Mathematics": "Mathematics", "Applied mathematics": "Mathematics", "Pure mathematics":
        "Computer Science": "Computer Sciences", "Computer Sciences": "Computer Sciences", "Alg
        "Information sciences": "Information sciences", "Information science": "Information sci
        "Earth Sciences": "Earth Sciences", "Earth Science": "Earth Sciences", "Atmospheric sci
        "Biology Science": "Biology Science", "Aerobiology": "Biology Science", "Bacteriology":
        "Physical sciences": "Physical sciences", "Physical science": "Physical sciences", "Acc
        "Chemical science": "Chemical sciences", "Chemical sciences": "Chemical sciences", "Ana

        # 2 Engineering and Technology
        "Civil engineering": "Civil engineering", "Architecture engineering": "Civil engineerin
        "Electrical, electronic and information engineering": "Electrical, electronic and infor
        "Mechanical engineering": "Mechanical engineering", "Applied mechanics": "Mechanical en
        "Aerospace engineering": "Aerospace engineering", "Aeronautical engineering": "Aerospac
        "Chemical engineering": "Chemical engineering", "Chemical engineering (plants, products
        "Materials engineering": "Materials engineering", "Ceramics": "Materials engineering",
        "Bioengineering and Biomedical engineering": "Bioengineering and Biomedical engineering
        "Environmental engineering": "Environmental engineering", "Energy and fuels": "Environm
        "Environmental biotechnology": "Environmental biotechnology", "Bioremediation": "Enviro
         "Industrial biotechnology":  "Industrial biotechnology", "Bio-derived novel materials"
         "Nano-technology":  "Nano-technology", "Nano-materials": "Nano-technology", "Nano-proc

        # 3 Medical and Health Sciences
        "Basic medicine": "Basic medicine", "Anatomy and morphology": "Basic medicine", "Human
        "Clinical medicine": "Clinical medicine", "Allergy": "Clinical medicine", "Anaesthesiol
        "Health science": "Health sciences", "Health sciences": "Health sciences", "Epidemiolog
        "Medical biotechnology": "Medical biotechnology", "Biomedical devices": "Medical biotec

        # 4 Agricultural Sciences
        "Agriculture, forestry, and fisheries": "Agriculture, forestry, and fisheries", "Agricu
        "Animal and dairy sciences": "Animal and dairy sciences", "Animal science": "Animal and
        "Veterinary sciences": "Veterinary sciences", "Veterinary anaesthesiology": "Veterinary
        "Agricultural biotechnology": "Agricultural biotechnology", "Biomass feedstock producti

        # 5 Social Sciences
        "Psychology": "Psychology", "Biological Psychology": "Psychology", "Clinical Psychology
        "Economics, finance and business": "Economics, finance and business", "Business and Mana
        "Educational sciences": "Educational sciences", "Educational science": "Educational sci
        "Sociology": "Sociology", "Anthropology": "Sociology", "Demography": "Sociology", "Ethn
        "Law": "Law", "Canon Law": "Law", "Civil Law": "Law", "Comparative Law": "Law", "Compet
        "Political sciences": "Political sciences", "Political science": "Political sciences",
        "Social and economic geography": "Social and economic geography", "Cultural and economi
        "Media and communications": "Media and communications", "Information science - social":

        # 6 "Humanities",
        "History and Archaeology": "History and Archaeology", "Archaeology": "History and Archa
        "Languages and literature": "Languages and literature", "General language studies": "La
        "Philosophy, ethics and religion": "Philosophy, ethics and religion", "Ethics": "Philos
        "Arts": "Arts", "Architectural design": "Arts", "Folklore studies": "Arts", "Media Stud

        # 7 "Support Activities"
        "Archives": "Support Activities", "Development": "Support Activities", "Urban planning"
}

def map_fos(mapper, count_mapper):
    def map_fos_(col):
        if col == "" or not col:
            return None
        fields = list(filter(None, [mapper.get(t) for t in col]))
        if len(fields):
            return fields[0]
        else:
            col_count = [count_mapper[x] for x in col]
            return col[col_count.index(max(col_count))]
    return udf(map_fos_, StringType())
```

```
def map_fos_id(rdd):
    def map_fos_id_(col):
        if col == "null" or col == "" or not col:
            return None
        try:
            matches = [fosTuple[0] for fosTuple in list(rdd.items()) if fosTuple[1] == col]
            if len(matches):
                return matches[0]
            else:
                return None
        except ValueError:
            return None
    return udf(map_fos_id_, LongType())
```

In [0]:
```
# Finding relevant `Field_of_Study` from `fos` list with mapped value with `translate` udf into
union = union.select("*", F.col("fos"), map_fos(decipline_mapper, count_mapper)("fos").alias("T
# Dropping `fos` column
union = union.drop("fos")

# Building Frame of distinct disciplines out of "Field_of_Study" column.
FoS_frame = distinct_frame_from_cols(union, ["Text"])
save_delta_frame(FoS_frame, "FieldOfStudy")

# Reading Mapping field of study to id, wuth RDD map for replacing "Field_of_Study" to relevant
FoSrdd = FoS_frame.rdd.collectAsMap()

union = union.withColumn("FOS_ID", map_fos_id(FoSrdd)("Text")).drop("Text")
```

Venue Table (Conference/Workshop where article was presneted/cited)

- Extract and flatten structure from main record
- Clean raw data and fetch Name, acronym and relevant url
- Remove Duplicates
- Save table

In [0]:
```
import requests

def venue_API(venue_string):
    if venue_string and venue_string != '':
        venue_string = venue_string.split(' ')[0]
        URL = "http://dblp.org/search/venue/api?q=" + venue_string + "%3A$&format=json"
        try:
            r = requests.get(url = URL)
            if r.status_code == 200:
                data = r.json()
                coAuths=[]
                joursConfs=[]
                data = data['result']['hits']
                if int(data['@total']) > 0:
                    return data['hit'][0]['info']['venue'], data['hit'][0]['info']['acronym'], 
        except:
            pass
    return None, None, None

schema = StructType([
    StructField("name", StringType(), True),
    StructField("acronym", StringType(), True),
    StructField("src", StringType(), True),
])

venue_query_udf = udf(venue_API, schema)

# Exploding a column returns a new row for each element in the given array or map type.
# For each item in the map/array of data it creates a copy of the row and with that element in 
# Here, We only select the exploded column, and so we only get row with author object in the ge

venue_frame = union.select("venue")

venue_frame = venue_frame.selectExpr("venue.*")

venue_frame = venue_frame.dropDuplicates(["_id"])
venue_frame = venue_frame.select("*", F.when(venue_frame.raw.isNotNull(), venue_query_udf(F.col
venue_frame = venue_frame.drop('name_d', 'raw', 'name_s', 'name', 'sid', 'issn', 'online_issn',
venue_frame = venue_frame.select("*", "query_results.*")
venue_frame = venue_frame.drop("query_results")

venue_frame.drop("all", subset=["name", "name_s", "url"])

save_delta_frame(venue_frame, "Venue", clean=True)

## TODO:
# 1. Pull more info before save
```

Author and Organization Tables

- Explode (with posexplode to get AuthorRank)author details from main record
- Extract and flatten Organization for each author
- Clean the data and split Name
- Remove Duplicates
- Save table

In [0]:
```
# !pip install geograpy3 nltk -q
```

In [0]:
```
#import geograpy
#import nltk
#nltk.download('punkt')
#nltk.download('averaged_perceptron_tagger')
#nltk.download('maxent_ne_chunker')
#nltk.download('words')

#str(geograpy.locateCity("Michigan"))
#geograpy.get_place_context(text="University of Michigan, USA")

#print(geograpy.get_place_context(text="University of Tartu, Estonia"))
```

In [0]:
```
# Extracting Authors from the dataset

# Exploding a column returns a new row for each element in the given array or map type.
# For each item in the map/array of data it creates a copy of the row and with that element in i
# Here, We only select the exploded column, and so we only get row with author object in the ge

union = union.select("*", F.posexplode("authors").alias("AuthorRank", "author")).drop("authors"
authors_frame = union.selectExpr("author.*")
authors_frame = authors_frame.dropDuplicates(["_id"])

# selectExpr Projects a set of SQL expressions and returns a new DataFrame. e.g. (authors['name
authors_frame = authors_frame.drop("org_zh", "orgs_zh", "orcid", "oid")
authors_frame.printSchema()
```

```
root
 |-- _id: string (nullable = true)
 |-- bio: string (nullable = true)
 |-- email: string (nullable = true)
 |-- gid: string (nullable = true)
 |-- name: string (nullable = true)
 |-- name_zh: string (nullable = true)
 |-- oid_zh: string (nullable = true)
 |-- org: string (nullable = true)
 |-- orgid: string (nullable = true)
 |-- orgs: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- sid: string (nullable = true)
 |-- position: string (nullable = true)
 |-- avatar: string (nullable = true)
 |-- homepage: string (nullable = true)
```

In [0]:
```
org_frame = authors_frame.select("_id", "org", "orgs").withColumnRenamed("_id", "Author_ID")
org_frame = org_frame.na.drop("all").distinct()
org_frame = org_frame.withColumn("Organization", F.when(F.col("org").isNotNull(), F.col("org"))

org_frame = distinct_frame_from_cols(org_frame, ["Organization", "Author_ID"])
save_delta_frame(org_frame, "Organization", clean=True)
```

In [0]:
```
# TODO:
# 1. Extract Org, Country and city for each ORG
```

In [0]:
```
def author_name(name):
    if name:
        name = name.split()
        if len(name) > 1:
            if len(name) == 1:
                return (name[0], None, None)
            return (name[0], ' '.join(name[1:-1]), name[-1])
    return None, None, None

author_name_schema = StructType([
    StructField("FirstName", StringType(), True),
    StructField("MiddleName", StringType(), True),
    StructField("LastName", StringType(), True),
])

author_name_udf = udf(author_name, author_name_schema)

authors_frame = authors_frame.select("*", author_name_udf("name").alias("author_name"))
authors_frame = authors_frame.select("*", "author_name.*")
authors_frame = authors_frame.drop("name", "author_name", "name_zh", "bio", "sid", "position",

authors_frame = save_delta_frame(authors_frame, "Author", clean=True)
```

In [0]:
```
union = union.withColumn('doc_type', F.when(union.venue.raw.contains("@"), 'workshop').when(uni
type_frame = distinct_frame_from_cols(union, ["doc_type"]).withColumnRenamed("doc_type", "Descr
type_rdd = type_frame.rdd.collectAsMap()

union = union.withColumn("Type_ID", map_fos_id(type_rdd)("doc_type"))
```

In [0]:
```
union = union.withColumn("venue", union.venue._id)
union = union.withColumn("author", union.author._id)

union = union.withColumnRenamed("_id", "Publication_ID")
union = union.withColumnRenamed("author", "Author_ID")
union = union.withColumnRenamed("venue", "Venue_ID")
union = union.withColumn("AuthorRank", F.col("AuthorRank") + F.lit(1)).drop("doc_type", "volume
```

Keyword Lookup (Partially Implemented*)

- Count keyword occurances to device a threshold for mapping
- Remove keywords below threshold
- Explode into fact table
- Extract distinct and map unique ID in fact table in place of exploded keyword
- Save table

In [0]:
```
# keyword_frame = union.select(F.explode_outer("keywords").alias("key_countmap"))
# key_countmap = keyword_frame.groupBy("key_countmap").count()
# key_countmap = key_countmap.rdd.map(lambda row: row.asDict(True))
# # union = union.drop("key_countmap")
# keyword_counts = key_countmap.collect()
# keyword_counter = {}
# for keyword_count in keyword_counts:
#     keyword_counter[keyword_count['key_countmap']] = keyword_count['count']

# keyword_counter
```

Saving Fact Table

In [0]:
```
save_delta_frame(union, "FactTable", clean=True)
```

LOAD

Loading saved Tables back

Future tasks: Update code for streaming write and read of data

In [0]:
```
language = spark.read.format('delta').load(f'{delta_dir}Language')
```

In [0]:
```
field_of_study = spark.read.format('delta').load(f'{delta_dir}FieldOfStudy')
```

In [0]:
```
publications = spark.read.format('delta').load(f'{delta_dir}Publication')
```

In [0]:
```
venues = spark.read.format('delta').load(f'{delta_dir}Venue')
```

In [0]:
```
authors = spark.read.format('delta').load(f'{delta_dir}Author')
```

In [0]:
```
organizations = spark.read.format('delta').load(f'{delta_dir}Organization')
```

In [0]:
```
factTable = spark.read.format('delta').load(f'{delta_dir}FactTable')
```

Operations

- H-Index Reference

In [ ]:
```
joined = factTable.join(publications, factTable.Publication_ID == publications._id).drop("_id")
```

In [0]:
```
countFrame = joined.groupBy("Author_ID").agg(
        F.sum("NumberOfCitations").alias("TotalCitations"),
        F.count("Publication_ID").alias("PaperCount"),
    ).select("Author_ID", "TotalCitations", "PaperCount")

countFrame = authors.join(countFrame, countFrame.Author_ID == authors._id).withColumn("Name", F
display(countFrame.orderBy(F.col("PaperCount").desc()))
```

Author_ID	TotalCitations	PaperCount	Name
54055927dabfae8faa5c5dfa	31436	1388	H. Poor
53f445bcdabfaee4dc7ce5dc	16330	1211	Mohamed-Slim Alouini
5489ba6bdabfae8a11fb46ec	13268	1165	Lajos Hanzo
5429fd93dabfae61d494cf5d	14794	1133	Wen Gao
5487fa09dabfaed7b5fa33e9	13644	1087	Victor Leung
53f47977dabfae8a6845b643	32389	1051	Philip Yu
542c458bdabfae2b4e1fb0c8	7397	1044	Hai Jin
53f4e24cdabfaefc1b77b3c4	4452	898	Leonard Barolli
5484e546dabfae9b4013320f	8675	891	Chin-Chen Chang
53f48bd2dabfaea7cd1cd0ec	10702	862	Witold Pedrycz

In [0]:
```
h_index_frame = joined.join(countFrame, joined.Author_ID == joined.Author_ID).drop(joined.Autho
h_index_frame = h_index_frame.groupBy("Author_ID").agg(
        F.min(F.when(h_index_frame.NumberOfCitations >= h_index_frame.PaperCount, h_index_frame
)
h_index_frame = authors.join(h_index_frame, h_index_frame.Author_ID == authors._id).withColumn(
display(h_index_frame.orderBy(F.col("HIndex").desc()))
```

Author_ID	HIndex	Name
53f4cadadabfaee57c780346	13	Diane Tang
53f495f3dabfaeb4c477b931	13	Vardy, A.
53f435bddabfaee0d9b6004b	13	Anukool Lakhina
53f438ccdabfaeecd69758b9	12	Raman Sarin
53f45331dabfaeb22f4f450f	12	Ossama Abdel-Hamid
53f43a43dabfaee0d9b88ae2	12	Geoff Hulten
53f44fe3dabfaee0d9bd9bbf	12	Wolf-Dietrich Weber
54328e2adabfaeb4c6a8cee5	12	F. DiCesare
53f38ef0dabfae4b34a48867	12	D.P. Palomar
53f44fe8dabfaefedbb38737	11	Petri Tanskanen

In [0]:
```
# Checking output correctness for Author 'Diane Tang' with ID: 53f4cadadabfaee57c780346
display(joined.filter(F.col('Author_ID') == '53f4cadadabfaee57c780346').select("publication_ID"
```

publication_ID	Author_ID	NumberOfCitations
53e9a84eb7602d9703196d26	53f4cadadabfaee57c780346	22
53e9aca1b7602d970367f5cd	53f4cadadabfaee57c780346	17
53e9b016b7602d9703a6d933	53f4cadadabfaee57c780346	46
53e9b91eb7602d9704502769	53f4cadadabfaee57c780346	222
53e99e71b7602d970273518b	53f4cadadabfaee57c780346	297
53e9bba1b7602d97047ea322	53f4cadadabfaee57c780346	13
53e9a5a8b7602d9702ed419e	53f4cadadabfaee57c780346	93
53e9a67bb7602d9702fac039	53f4cadadabfaee57c780346	46
53e99f94b7602d97028665cf	53f4cadadabfaee57c780346	39
53e9a3c1b7602d9702cd1e6b	53f4cadadabfaee57c780346	56

In [0]: