databricks Project 1

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.

```
# 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 into memory.
    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.")


# The returned array should have one object of FileInfo with size
=2568255035

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

Extracting Archive into json

1. Extracting 7zip file into json.

```
!pip install py7zr -q
```

```
import py7zr

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

```
dbutils.fs.ls("file:/databricks/driver/dblpv13.json")
```

2. 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)

```
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))), line))
fin.close()
fout.close()
```

3. 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.

```
%mkdir data
```

```
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 = []
```

4. Moving files to dbfs FileStore from instance storage, to make it available for later.

```
# 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")
```

```
# Copying all json parts into filestore.
dbutils.fs.cp("file:/databricks/driver/data/", "dbfs:/FileStore
/data/split_data", recurse = True)
```

Transform

Reading data from databricks Filestore into dataframes (Checkpoint after data load)

```
import uuid
from functools import reduce
import pyspark.sql.functions as F
from pyspark.sql.types import StructType, ArrayType, StringType, LongType,
StructField, IntegerType
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 from start again. If you updated the split value, ignore
this assertion error"
```

```
# 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_PART_{num}.json" for num in range(file_count)])
# 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)
 |-- 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: long (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: long (nullable = true)
```

```
# 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)
```

```
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}/_checkpoints/etl-from-
json").outputMode('append').start(delta_path)
    # 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

```
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

```
# 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']


# Building Publication Frame
publication_frame = union.select("_id", "title", "volume", "issue",
"abstract", "pdf", "isbn", "issn", "doi", "url",
cite_count(citation_counter)("_id").alias("NumberOfCitations"))
publication_frame = publication_frame.withColumn("issn",
F.when(publication_frame.issn.rlike("[1-9]*-[1-9]*"),
F.col("issn")).otherwise(None))
# Removing extracted fields from the main schema
union = union.drop("title", "abstract", "pdf", "isbn", "issn", "doi", "url",
"references", "page_start", "page_end", "n_citation")
```

```
# Saving the table
publication_frame = save_delta_frame(publication_frame, "Publication",
clean=True)


### 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.

First we built a mapper to generalize desciplines. Secondly, we count `Field of study` topics to determine significance and importance of each. 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.

Used This (https://confluence.egi.eu/display/EGIG/Scientific+Disciplines) link to build a map to replace the specific field to generalized descipline.

```
# 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": "Mathematics", "Statistics and probability":
"Mathematics",
    "Computer Science": "Computer Sciences", "Computer Sciences": "Computer
Sciences", "Algorithms": "Computer Sciences", "Artificial Intelligence
(expert systems, machine learning, robotics)": "Computer Sciences",
"Computer architecture": "Computer Sciences", "Computer communications":
"Computer Sciences", "Computer graphics": "Computer Sciences", "Computer
security and reliability": "Computer Sciences", "Data structures": "Computer
Sciences", "Distributed computing": "Computer Sciences", "Human-computer
interaction": "Computer Sciences", "Operating systems": "Computer Sciences",
"Parallel computing": "Computer Sciences", "Programming languages":
"Computer Sciences", "Quantum computing": "Computer Sciences", "Software
engineering": "Computer Sciences", "Theory of computation": "Computer
Sciences",
    "Information sciences": "Information sciences", "Information science":
"Information sciences", "Data management": "Information sciences", "Data
mining": "Information sciences", "Information retrieval": "Information
sciences", "Information management": "Information sciences", "Knowledge
management": "Information sciences", "Multimedia, hypermedia": "Information
sciences",
    "Earth Sciences": "Earth Sciences", "Earth Science": "Earth Sciences",
"Atmospheric science": "Earth Sciences", "Climate research": "Earth
Sciences", "Geochemistry": "Earth Sciences", "Geology": "Earth Sciences",
"Geophysics": "Earth Sciences", "Hydrology": "Earth Sciences", "Mineralogy":
"Earth Sciences", "Oceanography": "Earth Sciences", "Palaeontology": "Earth
Sciences", "Physical geography": "Earth Sciences", "Seismology": "Earth
Sciences", "Volcanology": "Earth Sciences",
    "Biology Science": "Biology Science", "Aerobiology": "Biology Science",
"Bacteriology": "Biology Science", "Behavioural biology": "Biology Science",
"Biochemistry and molecular biology": "Biology Science", "Biodiversity
conservation": "Biology Science", "Bioinformatics": "Biology Science",
"Biological rhythm": "Biology Science", "Biology": "Biology Science",
"Biophysics": "Biology Science", "Botany": "Biology Science", "Cell
biology": "Biology Science", "Computational biology": "Biology Science",
"Cryobiology": "Biology Science", "Developmental biology": "Biology
Science", "Ecology": "Biology Science", "Evolutionary biology": "Biology
Science", "Genetics and heredity": "Biology Science", "Marine and Freshwater
biology": "Biology Science", "Mathematical biology": "Biology Science",
"Microbiology": "Biology Science", "Mycology": "Biology Science", "Plant
science": "Biology Science", "Reproductive biology": "Biology Science",
"Structural biology": "Biology Science", "Taxonomy": "Biology Science",
"Theoretical biology": "Biology Science", "Thermal biology": "Biology
Science", "Virology": "Biology Science", "Zoology": "Biology Science",
    "Physical sciences": "Physical sciences", "Physical science": "Physical
sciences", "Accelerator physics": "Physical sciences", "Acoustics":
"Physical sciences", "Aerosol physics": "Physical sciences", "Astrobiology":
"Physical sciences", "Astronomy": "Physical sciences", "Astroparticle
physics": "Physical sciences", "Astrophysics": "Physical sciences",
"Atomic": "Physical sciences", "Chemical physics": "Physical sciences",
"Computational physics": "Physical sciences", "Condensed matter physics":
"Physical sciences", "Cryogenics": "Physical sciences", "Fluid Mechanics":
"Physical sciences", "Fusion": "Physical sciences", "High energy physics":
"Physical sciences", "Mathematical physics": "Physical sciences", "Medical
physics": "Physical sciences", "Molecular physics": "Physical sciences",
"Nuclear physics": "Physical sciences", "Optics": "Physical sciences",
"Particle physics": "Physical sciences", "Physics": "Physical sciences",
"Planetary science": "Physical sciences", "Plasma physics": "Physical
sciences", "Space science": "Physical sciences", "Quantum physics":
"Physical sciences",
    "Chemical science": "Chemical sciences", "Chemical sciences": "Chemical
sciences", "Analytical chemistry": "Chemical sciences", "Chemistry":
"Chemical sciences", "Colloid chemistry": "Chemical sciences",
"Computational chemistry": "Chemical sciences", "Electrochemistry":
"Chemical sciences", "Inorganic and nuclear chemistry": "Chemical sciences",
"Mathematical chemistry": "Chemical sciences", "Organic chemistry":
"Chemical sciences", "Physical chemistry": "Chemical sciences", "Polymer
science": "Chemical sciences",

    # 2 Engineering and Technology
    "Civil engineering": "Civil engineering", "Architecture engineering":
"Civil engineering", "Civil engineering": "Civil engineering", "Civil
Protection": "Civil engineering", "Construction/Structural engineering":
"Civil engineering", "Transport engineering": "Civil engineering",
    "Electrical, electronic and information engineering": "Electrical,
electronic and information engineering", "Communication engineering and
systems": "Electrical, electronic and information engineering", "Computer
hardware and architecture": "Electrical, electronic and information
engineering", "Electrical and electronic engineering": "Electrical,
electronic and information engineering", "Robotics, Automation and Control
Systems": "Electrical, electronic and information engineering",
    "Mechanical engineering": "Mechanical engineering", "Applied mechanics":
"Mechanical engineering", "Audio engineering": "Mechanical engineering",
"Nuclear related engineering": "Mechanical engineering", "Reliability
analysis": "Mechanical engineering", "Thermodynamics": "Mechanical
engineering",
    "Aerospace engineering": "Aerospace engineering", "Aeronautical
engineering": "Aerospace engineering", "Astronautical engineering":
"Aerospace engineering",
    "Chemical engineering": "Chemical engineering", "Chemical engineering
(plants, products)": "Chemical engineering", "Chemical process engineering":
"Chemical engineering",
    "Materials engineering": "Materials engineering", "Ceramics": "Materials
engineering", "Coating and films": "Materials engineering", "Composites":
"Materials engineering", "Paper and wood": "Materials engineering",
"Textiles": "Materials engineering",
    "Bioengineering and Biomedical engineering": "Bioengineering and
Biomedical engineering", "Bioengineering": "Bioengineering and Biomedical
engineering", "Biomedical engineering": "Bioengineering and Biomedical
engineering",
    "Environmental engineering": "Environmental engineering", "Energy and
fuels": "Environmental engineering", "Geological engineering":
"Environmental engineering", "Geotechnics": "Environmental engineering",
"Ocean engineering": "Environmental engineering", "Mining and mineral
processing": "Environmental engineering", "Petroleum engineering":
"Environmental engineering", "Remote sensing": "Environmental engineering",
"Sea vessels": "Environmental engineering",
    "Environmental biotechnology": "Environmental biotechnology",
"Bioremediation": "Environmental biotechnology", "Diagnostic
biotechnologies": "Environmental biotechnology",
     "Industrial biotechnology":  "Industrial biotechnology", "Bio-derived
novel materials": "Industrial biotechnology", "Biocatalysis": "Industrial
biotechnology", "Bioderived bulk and fine chemicals": "Industrial
biotechnology", "Biofuels": "Industrial biotechnology", "Biomaterials":
"Industrial biotechnology", "Bioprocessing technologies": "Industrial
biotechnology", "Bioproducts": "Industrial biotechnology", "Fermentation":
"Industrial biotechnology",
     "Nano-technology":  "Nano-technology", "Nano-materials": "Nano-
technology", "Nano-processes": "Nano-technology",

    # 3 Medical and Health Sciences
    "Basic medicine": "Basic medicine", "Anatomy and morphology": "Basic
medicine", "Human genetics": "Basic medicine", "Immunology": "Basic
medicine", "Medicinal chemistry": "Basic medicine", "Neuroscience": "Basic
medicine", "Pathology": "Basic medicine", "Pharmacology and pharmacy":
"Basic medicine", "Physiology": "Basic medicine", "Toxicology": "Basic
medicine",
    "Clinical medicine": "Clinical medicine", "Allergy": "Clinical
medicine", "Anaesthesiology": "Clinical medicine", "Andrology": "Clinical
medicine", "Cardiac and Cardiovascular systems": "Clinical medicine",
"Critical care/Emergency medicine": "Clinical medicine", "Dentistry, oral
surgery/medicine": "Clinical medicine", "Dermatology and venereal diseases":
"Clinical medicine", "Gastroenterology and hepatology": "Clinical medicine",
"General and internal medicine": "Clinical medicine", "Geriatrics and
gerontology": "Clinical medicine", "Hematology": "Clinical medicine",
"Integrative and Complementary medicine": "Clinical medicine", "Medical
imaging": "Clinical medicine", "Nuclear medicine": "Clinical medicine",
"Obstetrics and gynaecology": "Clinical medicine", "Oncology": "Clinical
medicine", "Ophthalmology": "Clinical medicine", "Optometry": "Clinical
medicine", "Orthopaedics": "Clinical medicine", "Otorhinolaryngolog":
"Clinical medicine", "Paediatrics": "Clinical medicine", "Peripheral
vascular disease": "Clinical medicine", "Psychiatry": "Clinical medicine",
"Radiology": "Clinical medicine", "Respiratory systems": "Clinical
medicine", "Rheumatology": "Clinical medicine", "Surgery": "Clinical
medicine", "Transplantation": "Clinical medicine", "Urology and nephrology":
"Clinical medicine",
    "Health science": "Health sciences", "Health sciences": "Health
sciences", "Epidemiology": "Health sciences", "Health care science and
services": "Health sciences", "Health policy and services": "Health
sciences", "Infectious diseases": "Health sciences", "Medical ethics":
"Health sciences", "Nursing": "Health sciences", "Nutrition and Dietetics":
"Health sciences", "Occupational health": "Health sciences", "Parasitology":
"Health sciences", "Public and environmental health": "Health sciences",
"Social biomedical science": "Health sciences", "Sport and fitness science":
"Health sciences", "Substance abuse": "Health sciences", "Tropical
medicine": "Health sciences",
    "Medical biotechnology": "Medical biotechnology", "Biomedical devices":
"Medical biotechnology", "Health-related biotechnology": "Medical
biotechnology", "Pharmaceutical biotechnology": "Medical biotechnology",
"Biotechnology and medical ethics": "Medical biotechnology", "Molecular
diagnostics": "Medical biotechnology", "Biophysical manipulation": "Medical
biotechnology", "Agricultural Sciences": "Medical biotechnology",

    # 4 Agricultural Sciences
    "Agriculture, forestry, and fisheries": "Agriculture, forestry, and
fisheries", "Agriculture": "Agriculture, forestry, and fisheries",
"Agronomy, plant breeding, plant protection": "Agriculture, forestry, and
fisheries", "Fishery": "Agriculture, forestry, and fisheries", "Forestry":
"Agriculture, forestry, and fisheries", "Horticulture and viticulture":
"Agriculture, forestry, and fisheries", "Soil science": "Agriculture,
forestry, and fisheries",
    "Animal and dairy sciences": "Animal and dairy sciences", "Animal
science": "Animal and dairy sciences", "Dairy science": "Animal and dairy
sciences", "Husbandry": "Animal and dairy sciences", "Pets": "Animal and
dairy sciences",
    "Veterinary sciences": "Veterinary sciences", "Veterinary
anaesthesiology": "Veterinary sciences", "Veterinary medicine": "Veterinary
sciences", "Veterinary ophthalmology": "Veterinary sciences", "Veterinary
pathobiology": "Veterinary sciences", "Veterinary radiology": "Veterinary
sciences", "Veterinary reproduction": "Veterinary sciences", "Veterinary
surgery": "Veterinary sciences",
    "Agricultural biotechnology": "Agricultural biotechnology", "Biomass
feedstock production tech": "Agricultural biotechnology", "Biopharming":
"Agricultural biotechnology", "Diagnostics": "Agricultural biotechnology",
"Food biotechnology": "Agricultural biotechnology", "GM technology (crops,
livestock)": "Agricultural biotechnology", "Livestock cloning":
"Agricultural biotechnology", "Marker assisted selection": "Agricultural
biotechnology",

    # 5 Social Sciences
    "Psychology": "Psychology", "Biological Psychology": "Psychology",
"Clinical Psychology": "Psychology", "Cognitive Psychology": "Psychology",
"Comparative Psychology": "Psychology", "Developmental Psychology":
"Psychology", "Educational and School Psychology": "Psychology",
"Evolutionary Psychology": "Psychology", "Industrial–organisational
Psychology": "Psychology", "Personality Psychology": "Psychology", "Positive
Psychology": "Psychology", "Social Psychology": "Psychology",
    "Economics, finance and business": "Economics, finance and business",
"Business and Management": "Economics, finance and business", "Economics and
Econometrics": "Economics, finance and business", "Finance": "Economics,
finance and business", "Industrial relations": "Economics, finance and
business",
    "Educational sciences": "Educational sciences", "Educational science":
"Educational sciences", "General Education": "Educational sciences",
"Special Education (learning disabilities)": "Educational sciences",
    "Sociology": "Sociology", "Anthropology": "Sociology", "Demography":
"Sociology", "Ethnology": "Sociology", "Family studies": "Sociology",
"Social issues": "Sociology", "Social work": "Sociology", "Sociology":
"Sociology", "Women's and gender studie": "Sociology",
    "Law": "Law", "Canon Law": "Law", "Civil Law": "Law", "Comparative Law":
"Law", "Competition Law": "Law", "Constitutional Law": "Law", "Criminal
Law": "Law", "Islamic Law": "Law", "Jewish Law": "Law", "Jurisprudence
(Philosophy of Law)": "Law",
    "Political sciences": "Political sciences", "Political science":
"Political sciences", "Comparative politics": "Political sciences",
"Empirical pata analysis": "Political sciences", "International relations":
"Political sciences", "Organisation theory": "Political sciences",
"Political economy": "Political sciences", "Political philosophy":
"Political sciences", "Public administration": "Political sciences",
"Theories of the state": "Political sciences",
    "Social and economic geography": "Social and economic geography",
"Cultural and economic geography": "Social and economic geography",
"Transport planning": "Social and economic geography", "Urban studies":
"Social and economic geography",
    "Media and communications": "Media and communications", "Information
science - social": "Media and communications", "Journalism": "Media and
communications", "Library science": "Media and communications", "Media and
socio-cultural communication": "Media and communications",

    # 6 "Humanities",
    "History and Archaeology": "History and Archaeology", "Archaeology":
"History and Archaeology", "History (Prehistory; Ancient; Modern world)":
"History and Archaeology",
    "Languages and literature": "Languages and literature", "General
language studies": "Languages and literature", "General literature studies":
"Languages and literature", "Linguistics": "Languages and literature",
"Literary theory": "Languages and literature", "Specific languages":
"Languages and literature", "Specific literatures": "Languages and
literature",
    "Philosophy, ethics and religion": "Philosophy, ethics and religion",
"Ethics": "Philosophy, ethics and religion", "Philosophy of
science/technology": "Philosophy, ethics and religion", "Philosophy":
"Philosophy, ethics and religion", "Religious studies": "Philosophy, ethics
and religion", "Theology": "Philosophy, ethics and religion",
    "Arts": "Arts", "Architectural design": "Arts", "Folklore studies":
"Arts", "Media Studies (Film, Radio, TV)": "Arts", "Musicology": "Arts",
"Performing arts studies": "Arts",

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

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())


# Finding relevant `Field_of_Study` from `fos` list with mapped value with
`translate` udf into "Field_of_Study" column.
union = union.select("*", F.col("fos"), map_fos(decipline_mapper,
count_mapper)("fos").alias("Text"))
# 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 ID in the union table.
FoSrdd = FoS_frame.rdd.collectAsMap()

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


# joined = union.join(FoS_frame, union.FOS_ID == FoS_frame.id,
how="left").drop("FOS_ID", "ID").withColumnRenamed("Text", "Field_of_Study")
# joined.show(10)
```

Extracting Venue (Conference/Workshop where article was presneted/cited) from the dataset

```
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'], data['hit'][0]['info']['url']
        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 new column.
# Here, We only select the exploded column, and so we only get row with
author object in the generated frame.

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("raw"))).alias("query_results"))
venue_frame = venue_frame.drop('name_d', 'raw', 'name_s', 'name', 'sid',
'issn', 'online_issn', 'publisher', 'type', 'src', 'raw_zh', 't')
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

```
# !pip install geograpy3 nltk -q
```

```
#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"))
```

```
# 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 new column.
# Here, We only select the exploded column, and so we only get row with
author object in the generated frame.

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', 'email'] => [authors.name, authors.email])
authors_frame = authors_frame.drop("org_zh", "orgs_zh", "orcid", "oid")
authors_frame.printSchema()
```

```
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")).otherwise(F.col("orgs")
[0])).select("Organization", "Author_ID")

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

```
# TODO:
# 1. Extract Org, Country and city for each ORG
# 2. Save Org Frame

save_delta_frame(org_frame, "Organization", clean=True)
```

```
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", "avatar", "homepage", "oid", "orcid", "oid_zh",
"orgs_zh", "orgs", "orgid", "org", "gid")

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

```
union = union.withColumn('doc_type', F.when(union.venue.raw.contains("@"),
'workshop').when(union.volume.isNotNull(),
'journal').when(union.issue.isNotNull(), 'journal').otherwise('conference'))
type_frame = distinct_frame_from_cols(union,
["doc_type"]).withColumnRenamed("doc_type", "Description")
type_rdd = type_frame.rdd.collectAsMap()

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


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", "issue")
```

Keyword Lookup

```
# 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

```
save_delta_frame(union, "FactTable", clean=True)
```

LOAD

Loading saved frames

```
language = spark.read.format('delta').load(f'{delta_dir}Language')
language.count()
```

```
Out[4]: 1
```

```
field_of_study = spark.read.format('delta').load(f'{delta_dir}FieldOfStudy')
field_of_study.count()
```

```
Out[5]: 250
```

```
publications = spark.read.format('delta').load(f'{delta_dir}Publication')
publications.count()
```

```
Out[6]: 228801
```

```
venues = spark.read.format('delta').load(f'{delta_dir}Venue')
venues.count()
```

```
authors = spark.read.format('delta').load(f'{delta_dir}Author')
authors.count()
```

```
Out[7]: 376939
```

```
organizations = spark.read.format('delta').load(f'{delta_dir}Organization')
organizations.count()
```

```
Out[8]: 516834
```

```
factTable = spark.read.format('delta').load(f'{delta_dir}FactTable')
factTable.count()
```

```
Out[9]: 609686
```

```
factTable.printSchema()
```

```
root
 |-- Publication_ID: string (nullable = true)
 |-- keywords: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- Venue_ID: string (nullable = true)
 |-- year: long (nullable = true)
 |-- Lang_ID: long (nullable = true)
 |-- FOS_ID: long (nullable = true)
 |-- AuthorRank: integer (nullable = true)
 |-- Author_ID: string (nullable = true)
 |-- Type_ID: long (nullable = true)
```

Operations

- H-Index Reference (https://docs.microsoft.com/en-us/academic-services/graph/tutorial-databricks-hindex)

```
joined = factTable.join(publications, factTable.Publication_ID ==
publications._id).drop("_id").withColumn("NumberOfCitations",
F.col("NumberOfCitations").cast('int'))
joined.show(10)
```

```
+--------------------+--------------------+--------------------+----+------
-+------+----------+--------------------+-------+--------------------+-----
-+-----+--------------------+--------------------+------------+----+------
--------------+--------------------+-----------------+
|      Publication_ID|            keywords|            Venue_ID|year|Lang_I
D|FOS_ID|AuthorRank|           Author_ID|Type_ID|               title|volum
e|issue|            abstract|                 pdf|        isbn|issn|
doi|                 url|NumberOfCitations|
+--------------------+--------------------+--------------------+----+------
-+------+----------+--------------------+-------+--------------------+-----
-+-----+--------------------+--------------------+------------+----+------
--------------+--------------------+-----------------+
|53e99792b7602d970...|[feedback, feedfo...|555036c77cea80f95...|2008|   nul
l|  null|         1|53f468e9dabfaeb22...|   null|Positive Feedback...|    5
5|   10|                    |                null|            |null|10.110
9/TCSI.2008...|[http://dx.doi.or...|             null|
|53e99792b7602d970...|[integrated hse e...|                null|2006|   nul
l|  null|         1|53f442b6dabfaeee2...|   null|Requirements anal...|
|     |Human-Sensibility...|                null|            |null|10.1007
/978-3-540...|[http://dx.doi.or...|             null|
|53e99792b7602d970...|[integrated hse e...|                null|2006|   nul
l|  null|         3|53f431a4dabfaeb2a...|   null|Requirements anal...|
|     |Human-Sensibility...|                null|            |null|10.1007
/978-3-540...|[http://dx.doi.or...|             null|
|53e99792b7602d970...|[integrated hse e...|                null|2006|   nul
l|  null|         2|5405464bdabfae8fa...|   null|Requirements anal...|
|     |Human-Sensibility...|                null|            |null|10.1007
/978-3-540...|[http://dx.doi.or...|             null|
|53e99792b7602d970...|[0.35 micron, ota...|555037ab7cea80f95...|2006|   nul
l|  null|         4|53f45804dabfaeee2...|   null|A 140-dB CMRR Low...|
|     |This paper presen...|                null|1-4244-0387-1|null|10.1109
/APCCAS.20...|[http://dx.doi.or...|             null|
|53e99792b7602d970...|[0.35 micron, ota...|555037ab7cea80f95...|2006|   nul
l|  null|         3|53f4593adabfaec09...|   null|A 140-dB CMRR Low...|
|     |This paper presen...|                null|1-4244-0387-1|null|10.1109
/APCCAS.20...|[http://dx.doi.or...|             null|
|53e99792b7602d970...|[0.35 micron, ota...|555037ab7cea80f95...|2006|   nul
l|  null|         2|54891d6adabfae9b4...|   null|A 140-dB CMRR Low...|
|     |This paper presen...|                null|1-4244-0387-1|null|10.1109
/APCCAS.20...|[http://dx.doi.or...|            null|
|53e99792b7602d970...|[0.35 micron, ota...|555037ab7cea80f95...|2006|   nul
l|  null|         1|5484bc1fdabfae9b4...|   null|A 140-dB CMRR Low...|
|     |This paper presen...|                null|1-4244-0387-1|null|10.1109
/APCCAS.20...|[http://dx.doi.or...|            null|
|53e99792b7602d970...|[development envi...|53a7261320f7420be...|1993|   nul
l|  null|         2|53f35233dabfae4b3...|   null|A Prototyping and...|
|     |                    |//static.aminer.o...|             |null|
|                  []|             null|
|53e99792b7602d970...|[development envi...|53a7261320f7420be...|1993|   nul
l|  null|         3|53f4396fdabfaefed...|   null|A Prototyping and...|
|     |                    |//static.aminer.o...|             |null|
|                  []|             null|
+--------------------+--------------------+--------------------+----+------
-+------+----------+--------------------+-------+--------------------+-----
-+-----+--------------------+--------------------+-------------+----+------
--------------+--------------------+-----------------+
only showing top 10 rows
```

```
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.concat("FirstName", F.lit(" "),
"LastName")).drop("_id", "email", "FirstName", "MiddleName", "LastName")
display(countFrame.orderBy(F.col("PaperCount").desc()))
```

	Author_ID	TotalCitations	PaperCount	Name
1	53f48abedabfaea6fb77b490	185	87	Thomas Huang
2	53f483a0dabfaeb1a7cd15ce	45	60	Ajith Abraham
3	5429fd93dabfae61d494cf5d	39	59	Wen Gao
4	548a0c6ddabfae9b40134ec5	115	58	N. Alon
5	53f4b415dabfaed31c77b3ba	183	57	Moshe Vardi
6	53f48046dabfae963d259326	501	55	Anil Jain

Truncated results, showing first 1000 rows.

```
h_index_frame = joined.join(countFrame, joined.Author_ID ==
joined.Author_ID).drop(joined.Author_ID)
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.PaperCount).otherwise(0)).alias('HIndex')
)
h_index_frame = authors.join(h_index_frame, h_index_frame.Author_ID ==
authors._id).withColumn("Name", F.concat("FirstName", F.lit(" "),
"LastName")).drop("_id", "email", "FirstName", "MiddleName", "LastName")
display(h_index_frame.orderBy(F.col("HIndex").desc()))
```

	Author_ID	HIndex	Name
1	53f438a6dabfaec09f1925de	4	Dan Frankowski
2	53f43108dabfaeb22f4369b0	4	Eitan Sharon
3	53f42b70dabfaeb22f3e99da	3	David Nistér
4	54863528dabfaed7b5fa2852	3	David Tse
5	53f4c86ddabfaedce56646aa	3	Francisco Barahona
6	53f438c8dabfaedd74db6e84	3	claude kirchner

Truncated results, showing first 10000 rows.

	publication_ID	Author_ID	NumberOfCitations
1	53e999d8b7602d970221cacf	53f43108dabfaeb22f4369b0	8
2	53e99aacb7602d9702328daa	53f43108dabfaeb22f4369b0	12
3	53e999d2b7602d970221765b	53f43108dabfaeb22f4369b0	10
4	53e999bbb7602d97022023d0	53f43108dabfaeb22f4369b0	11

Showing all 4 rows.