Data Design and Modelling – Project Work Part 3

Claudio Maggioni, Lorenzo Martignetti

December 18, 2022

[1]:
```
# Import the basic spark library
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, FloatType,␣
 ↪ArrayType, IntegerType, DateType
from pyspark.sql import functions as F
from pyspark.sql.types import StringType
from pyspark.sql.functions import explode, col, sum, avg, count, max, first,␣
 ↪last

from faker import Factory
import random

from datetime import datetime
from time import time
```

[2]:
```
# Create an entry point to the PySpark Application
spark = SparkSession.builder \
      .master("local") \
      .appName("DDMProjectWorkPart3") \
      .getOrCreate()
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
```

```
22/12/18 17:40:40 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform… using builtin-java classes where applicable
```

0.1 Schema definition and faker setup

Here we define a series of Spark user-defined functions (or UDFs) to account for the missing fields in the DBLP dataset. We generate all these fields with the *Faker* library, namely:

- The `Content` field for article records;
- The `Bio`, `Email` and `Affiliation` fields for authors. Note that we generate affiliation using a pool of 500 random university names, generated by appending the string `"University of "` to a random town name (the same strategy used for Project Work Part 2).

We define more UDFs for the references dataframe, however we declare them next to the references import code.

```
[3]: fake = Factory.create()

# Article
fake_content_udf = F.udf(lambda: fake.paragraph(nb_sentences=25), StringType())

# Author
fake_unis = []
for i in range(0, 500):
    (_, _, loc, _, _) = fake.location_on_land()
    fake_unis.append("University of " + loc)

fake_bio_udf = F.udf(lambda: fake.paragraph(nb_sentences=5), StringType())
fake_email_udf = F.udf(lambda: fake.ascii_email(), StringType())
fake_affiliation_udf = F.udf(lambda: fake_unis[random.randint(0, len(fake_unis)␣
 ↪- 1)], StringType())
```

Here we define the schema for all dataframes we import. This is ostensibly for documentation, as the schema is automatically defined during the import process thanks to the `inferSchema` Spark CSV reader option.

Note that `article` and `author` are in a N-to-N relation, with `article_author` acting as a join table. Additionally, N `article` rows may reference a `journal` and N `reference` rows may reference an `article`.

```
[4]: article_schema = StructType([ \
    StructField("ID", IntegerType(), False), \
    StructField("Title", StringType(), False), \
    StructField("Content", StringType(), True), \
    StructField("MetadataDate", DateType(), True), \
    StructField("JournalId", IntegerType(), True), \
    StructField("Volume", StringType(), True), \
    StructField("NumberInVolume", StringType(), True), \
    StructField("PagesInVolume", StringType(), True), \
    StructField("ObjectIds", ArrayType(StringType()), True)
])

article_author_schema = StructType([ \
    StructField("ArticleId", IntegerType(), False), \
    StructField("AuthorId", IntegerType(), False)
])

author_schema = StructType([ \
    StructField("ID", IntegerType(), False), \
    StructField("Name", StringType(), False), \
    StructField("Email", StringType(), True), \
    StructField("Affiliation", DateType(), True), \
    StructField("Bio", IntegerType(), True)
])
```

```
journal_schema = StructType([ \
    StructField("ID", IntegerType(), False), \
    StructField("Name", StringType(), False)
])

reference_schema = StructType([ \
    StructField("ID", IntegerType(), False), \
    StructField("ArticleId", IntegerType(), False), \
    StructField("RefNumber", IntegerType(), True), \
    StructField("InternalPaperId", IntegerType(), True), \
    StructField("Title", StringType(), True), \
    StructField("Authors", ArrayType(StringType()), True), \
    StructField("Journal", StringType(), True), \
    StructField("JournalId", IntegerType(), True), \
    StructField("Volume", StringType(), True), \
    StructField("NumberInVolume", StringType(), True)
])
```

0.2 The DBLP dataset

Like in Project Work Part 1, we choose to use the XML dataset of the DBLP scientific publication directory. We use the script in the ThomHurks/dblp-to-csv GitHub repo to generate CSV files from the downloaded XML dump. For the scope of this part we only consider the publications of type “article”.

We use the `--relations` option of the *dblp-to-csv* script to extract author and journal names into separate CSV files. The script also generates IDs for all entities (articles, authors and journals), and two files akin to join tables in a relational model to combine article rows with the extracted author and journal rows.

The following list of shell commands downloads the DBLP dataset, the *dblp-to-csv* script and generates the CSV files according to the specification required by this notebook:

```
curl -o dblp.xml.gz https://dblp.org/xml/dblp.xml.gz
gunzip dblp.xml.gz

# download the DTD specification of the DBLP XML format
curl -o dblp.dtd https://dblp.org/xml/dblp.dtd

git clone https://github.com/ThomHurks/dblp-to-csv

mkdir csv-import

dblp-to-csv/XMLToCSV.py --annotate dblp.xml dblp.dtd dblp_csv.csv \
    --relations journal:article_journal author:article_author

for t in article; do
    tr ';' '\n' <dblp_csv_${t}_header.csv | sed 's/:.*//g' | \
        tr '\n' ';' | awk 1 | cat - dblp_csv_${t}.csv | \
        sed -E 's/\{?\\""\}?/""/g' > csv-import/${t}.csv;
done

cp dblp_csv_{author|journal}_* dblp_csv_{author|journal}.csv csv-import
```

0.3 Data import and dataframes creation

Here we create the `article` dataframe. To create it we mainly need to read the article CSV file and join it with the article-to-journal join table created by *dblp-to-csv* script, which we can do while preserving 1NF since we know that an article may be present only in a single journal.

For the scope of this assignment we decide to fix the number of articles to 10000, selecting them at random from the export.

[5]:
```
articles_path = "./csv-import/article.csv"
articles_journals_path = "./csv-import/dblp_csv_journal_article_journal.csv"

df_articles_csv = spark.read \
    .option("delimiter", ";") \
    .option("header", True) \
    .csv(articles_path, inferSchema=True) \
    .select("id", "title", "mdate", "volume", "number", "pages", "ee") \
    .where("id > 2") \
    .withColumn('Content', fake_content_udf()) \
    .withColumnRenamed("id", "ID") \
    .withColumnRenamed("title", "Title") \
    .withColumnRenamed("mdate", "MetadataDate") \
    .withColumnRenamed("volume", "Volume") \
    .withColumnRenamed("number", "NumberInVolume") \
    .withColumnRenamed("pages", "PagesInVolume") \
    .withColumnRenamed("ee", "ObjectIds") \
    .na.drop(subset=["ID", "Title"]) \
    .withColumn("ID", F.coalesce(F.col("ID"), F.lit(0))) \
    .withColumn("Title", F.coalesce(F.col("Title"), F.lit(""))) \
    .select("ID", "Title", "MetadataDate", "Volume", "NumberInVolume",␣
 ↪"PagesInVolume", \
            F.split("ObjectIds", "\\|").alias("ObjectIds"))

# force ID and Title to be not-nullable, given the .na.drop coalescing never␣
 ↪actually happens
# https://stackoverflow.com/a/58515908

df_article_journals_csv = spark.read \
    .option("delimiter", ";") \
    .option("header", True) \
    .csv(articles_journals_path, inferSchema=True) \
    .withColumnRenamed(":END_ID", "JournalId")
```

```
df_articles = df_articles_csv \
    .join(df_article_journals_csv, \
          df_article_journals_csv[':START_ID'] == df_articles_csv.ID, 'left') \
    .drop(":START_ID") \
    .orderBy(F.rand()).limit(10000)

df_articles.printSchema()
df_articles.show()
```

```
root
 |-- ID: integer (nullable = false)
 |-- Title: string (nullable = false)
 |-- MetadataDate: timestamp (nullable = true)
 |-- Volume: string (nullable = true)
 |-- NumberInVolume: string (nullable = true)
 |-- PagesInVolume: string (nullable = true)
 |-- ObjectIds: array (nullable = true)
 |    |-- element: string (containsNull = false)
 |-- JournalId: integer (nullable = true)

[Stage 8:==========================================>                (5 + 1) / 7]

+-------+--------------------+-------------------+--------------+--------------
+-------------+--------------------+---------+
|     ID|               Title|       MetadataDate|
Volume|NumberInVolume|PagesInVolume|           ObjectIds|JournalId|
+-------+--------------------+-------------------+--------------+--------------
+-------------+--------------------+---------+
|7476125|Offline Drawing o…|2018-08-13 00:00:00|abs/1608.08385|          null|
null|[http://arxiv.org…| 12791182|
|7798842|Reasoning about S…|2020-09-02 00:00:00|            22|             2|
241-262|[https://doi.org/…| 12791281|
|9477159|System Dynamics M…|2021-10-14 00:00:00|            11|             5|
677|[https://doi.org/…| 12792549|
|7286983|Wireless Transmis…|2018-08-13 00:00:00|abs/1805.09923|          null|
null|[http://arxiv.org…| 12791182|
|7316024|Distributed Struc…|2018-08-13 00:00:00| abs/1207.1345|          null|
null|[http://arxiv.org…| 12791182|
|7863079|A Novel Human Aut…|2018-11-14 00:00:00|            12|             6|
7828-7854|[https://doi.org/…| 12791310|
|6720476|Biogenetic Temper…|2021-10-14 00:00:00|            11|             6|
735-737|[https://doi.org/…| 12790859|
|7912232|Routing of multip…|2020-04-02 00:00:00|             6|             9|
1617-1622|[https://doi.org/…| 12791324|
|9296802|The Third Version…|2021-10-14 00:00:00|             5|             4|
null|[https://doi.org/…| 12792370|
|7021903|Demand for housin…|2020-09-17 00:00:00|            18|             1|
61-68|[https://doi.org/…| 12791062|
|8624192|Visualization of …|2018-11-14 00:00:00|            18|            12|
2198-2207|[http://doi.ieeec…| 12791925|
|9028185|Network Topology …|2020-05-20 00:00:00|            56|            10|
2262-2275|[https://doi.org/…| 12792144|
|8756232|Modelling, analys…|2020-07-03 00:00:00|            12|             3|
207-216|[https://doi.org/…| 12791998|
|7841131|Person re-identif…|2020-01-15 00:00:00|            97|          null|
null|[https://doi.org/…| 12791303|
|8571383|Social media and …|2022-01-03 00:00:00|            39|          null|
404-412|[https://doi.org/…| 12791876|
|9211872|Reclaiming Spare …|2020-03-13 00:00:00|          2011|          null|
null|[https://doi.org/…| 12792282|
|7654415|Towards Automatic…|2019-10-02 00:00:00|abs/1909.13184|          null|
null|[http://arxiv.org…| 12791182|
|8165836|Smart Real Time A…|2021-02-26 00:00:00|            20|             3|
347-364|[https://doi.org/…| 12791533|
|8549734|Large-scale image…|2020-05-11 00:00:00|            79|         13-14|
9663|[https://doi.org/…| 12791859|
|7535983|On Coreset Constr…|2018-08-13 00:00:00|abs/1612.07516|          null|
null|[http://arxiv.org…| 12791182|
+-------+--------------------+-------------------+--------------+--------------
+-------------+--------------------+---------+
only showing top 20 rows
```

Here we import the `article-to-author` CSV file as a dataframe. We filter its rows to make sure we only include the articles we selected at random from the previous step from performance reasons.

Note the use of the `F.coalesce` function to coalesce `ArticleId` and `AuthorId` with 0. As we know that the columns are never null by analyzing the source data, this is simply a workaround to force the schema of the resulting DataFrame to mark these columns as not-null.

[100]:
```python
articles_authors_path = "./csv-import/dblp_csv_author_article_author.csv"

df_article_ids = df_articles.select(F.col("ID").alias("ArtId"))

df_articles_authors = spark.read \
    .option("delimiter", ";") \
    .option("header", True) \
    .csv(articles_authors_path, inferSchema=True) \
    .withColumnRenamed(":START_ID", "ArticleId") \
    .withColumnRenamed(":END_ID", "AuthorId") \
    .withColumn("ArticleId", F.coalesce(F.col("ArticleId"), F.lit(0))) \
    .withColumn("AuthorId", F.coalesce(F.col("AuthorId"), F.lit(0))) \
    .join(df_article_ids, df_article_ids["ArtId"] == F.col("ArticleId"),␣
↪'inner') \
    .select("ArticleId", "AuthorId") \
    .distinct()

df_articles_authors.printSchema()
df_articles_authors.show()
```

```
root
 |-- ArticleId: integer (nullable = false)
 |-- AuthorId: integer (nullable = false)

[Stage 1578:>                                                       (0 + 1) / 1]

+---------+--------+
|ArticleId|AuthorId|
+---------+--------+
|  6636860| 9567664|
|  6636860| 9570761|
|  7374783| 9573169|
|  7693829| 9601657|
|  6555847| 9608005|
|  7374783| 9626452|
|  7946825| 9631289|
|  8139540| 9636064|
|  7565583| 9636102|
|  9321630| 9644172|
|  7668271| 9650410|
|  7668271| 9651402|
|  7371049| 9658197|
|  8037829| 9665927|
|  8037829| 9667969|
|  8170953| 9671694|
|  6636860| 9672057|
|  6649339| 9673588|
|  7324396| 9674337|
|  8362140| 9674444|
+---------+--------+
only showing top 20 rows
```

Here we import the `authors`. We perform a similar filter process to the aforementioned one, dropping the authors that did not publish any article that we randomly selected for performance reasons.

[7]:
```python
authors_path = "./csv-import/dblp_csv_author.csv"

df_author_ids = df_articles_authors.select(F.col("AuthorId").alias("AutId"))

# More info on coalescing: https://youtu.be/ysPtBjY8o_A
df_authors = spark.read \
    .option("delimiter", ";") \
    .option("header", True) \
    .csv(authors_path, inferSchema=True) \
    .withColumnRenamed(":ID", "ID") \
    .withColumnRenamed("author:string", "Name") \
    .join(df_author_ids, df_author_ids["AutId"] == F.col("ID"), 'inner') \
    .select("ID", "Name") \
    .distinct() \
    .withColumn("ID", F.coalesce(F.col("ID"), F.lit(0))) \
    .withColumn("Name", F.coalesce(F.col("Name"), F.lit(""))) \
    .withColumn('Email', fake_email_udf()) \
    .withColumn('Affiliation', fake_affiliation_udf()) \
    .withColumn('Bio', fake_bio_udf())

df_authors.printSchema()
df_authors.show()
```

```
root
 |-- ID: integer (nullable = false)
 |-- Name: string (nullable = false)
 |-- Email: string (nullable = true)
 |-- Affiliation: string (nullable = true)
 |-- Bio: string (nullable = true)

[Stage 32:>                                                         (0 + 1) / 1]

+--------+--------------------+--------------------+--------------------+--------
------------+
|      ID|                Name|               Email|         Affiliation|
Bio|
+--------+--------------------+--------------------+--------------------+--------
------------+
|9566872|          Adnan Abid|wjohnson@clark-mc…|University of Vernon|Already
sell deci…|
|9567627|       Roberto Rossi|hmontgomery@hotma…|   University of Biu|Seven
exist risk …|
|9569592|  Hannes Hartenstein|robertsbeth@ramir…|University of Losser|Card
data with be…|
|9572255|     Bülent Karasözen| joshua34@barber.net| University of Kulai|Ready
effort duri…|
|9608965|        Xiaowei Zhao|timothyvazquez@gm…|University of Par…|Build
wall push c…|
|9638842|              Li Yan|loganjohnson@gmai…|University of Artsyz|School
eye about…|
|9641092|       Karen Livescu|  dyoung@hotmail.com|University of Eas…|Some up
no. Follo…|
|9641772|    Edna Dias Canedo|  cindy01@rivera.com|University of Sas…|Consider
parent l…|
|9652591|     Petrik Clarberg|harristiffany@gil…|University of
Lan…|Establish black m…|
|9653203|         Jan Jerabek|zrodriguez@black-…| University of Tavda|Deep
event hold w…|
|9666198|Mihaela van der S…|hvelasquez@hotmai…|University of Voz…|Study
foot over a…|
|9666973|      Esther Pacitti|edwardmelton@hotm…| University of
Tonga|Represent respons…|
|9667290|      Yannis Kotidis|    paul36@yahoo.com|University of Glo…|Wish
special guy…|
|9669347|           Ove Frank|   pmartin@yahoo.com|University of New…|Art
other radio e…|
|9669369|   Weldon A. Lodwick|hughesrodney@vill…|University of Bel…|Exactly
relate se…|
|9676721|       Glen Takahara|jenniferbaker@hot…|University of Bru…|Catch
still follo…|
|9734805|      Hibiki Tsukada|christinahensley@…|University of Aklera|Success
economic …|
|9749941|     Tobias Gemaßmer|brownjohn@hotmail…|University of Agu…|Lot
conference ca…|
|9819186|         Tarak Nandy|kevinduncan@hotma…| University of Lille|Election
environm…|
|9857863|      Jun Zhang 0024|perezolivia@hotma…|University of Kan…|Chair
pretty rece…|
+-------+--------------------+--------------------+--------------------+--------
------------+
only showing top 20 rows
```

Finally, here we import the `journals`. Note that here we actually import journals that may have no article in them out of the randomly selected articles. As this does not affect the queries or performance, we simply ignore this.

[8]:
```
journals_path = "./csv-import/dblp_csv_journal.csv"

df_journals = spark.read \
    .option("delimiter", ";") \
    .option("header", True) \
    .csv(journals_path, inferSchema=True) \
    .withColumnRenamed(":ID", "ID") \
    .withColumnRenamed("journal:string", "Name") \
    .withColumn("ID", F.coalesce(F.col("ID"), F.lit(0))) \
    .withColumn("Name", F.coalesce(F.col("Name"), F.lit("")))

df_journals.printSchema()
df_journals.show()
```

```
root
 |-- ID: integer (nullable = false)
 |-- Name: string (nullable = false)

+--------+--------------------+
|      ID|                Name|
+--------+--------------------+
|12790675|      World Wide Web|
|12790676|         SIGMOD Rec.|
|12790677|       SIGMOD Record|
|12790678|EAI Endorsed Tran…|
|12790679|Int. J. Trust. Ma…|
|12790680|Expert Syst. Appl. X|
|12790681|Bull. dInformatiq…|
|12790682|Trans. Large Scal…|
|12790683| Stud. Inform. Univ.|
|12790684|IEEE Intell. Tran…|
|12790685|    Control. Cybern.|
|12790686|Syst. Control. Lett.|
|12790687|Sci. Comput. Prog…|
|12790688|Found. Comput. Math.|
|12790689| Theor. Comput. Sci.|
|12790690|  Inf. Technol. Dev.|
|12790691|EURASIP J. Adv. S…|
|12790692|IEEE Embed. Syst…|
|12790693|J. Assoc. Inf. Syst.|
|12790694|EAI Endorsed Tran…|
+--------+--------------------+
only showing top 20 rows
```

The DBLP XML dataset has no **reference** data, therefore we generate our own with the aid of UDF functions using Faker. For each article we generate 1 to 15 random references with may either point to an article in our dataset or to an external article.

[9]:
```
reference_to_explode_schema = ArrayType(StructType([ \
    StructField("RefNumber", IntegerType(), True), \
    StructField("InternalPaperId", IntegerType(), True), \
    StructField("Title", StringType(), True), \
    StructField("Authors", ArrayType(StringType()), True), \
    StructField("Journal", StringType(), True), \
    StructField("JournalId", IntegerType(), True), \
    StructField("Volume", StringType(), True), \
    StructField("NumberInVolume", StringType(), True)
]))

# Pool of articles to randomly cite
# We can't select them ad-hoc in the UDF function as DataFrames cannot be␣
 ↪pickled
# See: https://stackoverflow.com/q/47249292
articles: list[dict] = spark.read \
    .option("delimiter", ";") \
    .option("header", True) \
    .csv(articles_path, inferSchema=True) \
    .alias("art1") \
    .join(df_articles, F.col("art1.ID") == df_articles_csv["ID"], 'inner') \
    .select("art1.*") \
    .select("id", "title", "volume", "number", "pages", "journal",
            F.split(F.col("author"), '\|', -1).alias('author')) \
    .withColumn("id", F.coalesce(F.col("id"), F.lit(0))) \
    .withColumn("title", F.coalesce(F.col("title"), F.lit(""))) \
    .alias("art2") \
    .join(df_article_journals_csv, df_article_journals_csv[':START_ID'] == F.
 ↪col("art2.id"), 'left') \
    .drop(":START_ID") \
    .collect()

def internal_fake_reference(ref: int):
    # Select Random article
    article: dict = articles[random.randint(0, len(articles) - 1)]

    return {
        "RefNumber": ref,
        "InternalPaperId": article["id"],
        "Title": article["title"],
        "Authors": article["author"],
        "Journal": article["journal"],
        "JournalId": article["JournalId"],
        "Volume": article["volume"],
        "NumberInVolume": article["number"]
    }

def external_fake_reference(ref: int):
    return {
        "RefNumber": ref,
        "InternalPaperId": None,
        "Title": fake.catch_phrase() + ": " + fake.catch_phrase(),
        "Authors": [fake.name() for x in range(random.randint(1, 6))],
        "Journal": fake.bs(),
        "JournalId": None,
        "Volume": random.randint(1, 500),
        "NumberInVolume": random.randint(1, 200)
    }

def rand_bool() -> bool:
    return bool(random.getrandbits(1))

def rand_references(min_count: int, max_count: int) -> list[dict]:
    return [internal_fake_reference(x + 1) if rand_bool() else␣
 ↪external_fake_reference(x + 1) \
            for x in range(random.randint(min_count, max_count))]

fake_references_udf = F.udf(lambda : rand_references(1, 15),␣
 ↪reference_to_explode_schema)

df_references = df_articles \
    .withColumn("explode", fake_references_udf()) \
    .withColumnRenamed("ID", "ArticleId") \
    .select("ArticleId", F.explode("explode").alias("explode")) \
    .select("explode.*", "ArticleId") \
    .withColumn("ID", F.monotonically_increasing_id())

df_references.printSchema()
df_references.show()
```

```
root
 |-- RefNumber: integer (nullable = true)
 |-- InternalPaperId: integer (nullable = true)
 |-- Title: string (nullable = true)
 |-- Authors: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- Journal: string (nullable = true)
 |-- JournalId: integer (nullable = true)
 |-- Volume: string (nullable = true)
 |-- NumberInVolume: string (nullable = true)
 |-- ArticleId: integer (nullable = false)
 |-- ID: long (nullable = false)

[Stage 52:>                                                         (0 + 1) / 1]

+---------+---------------+--------------------+--------------------+-----------
---------+---------+--------------+--------------+---------+---+
|RefNumber|InternalPaperId|               Title|             Authors|
Journal|JournalId|       Volume|NumberInVolume|ArticleId| ID|
+---------+---------------+--------------------+--------------------+-----------
---------+---------+-------------+-------------+---------+---+
|        1|        7490681|UNIFUZZ: A Holist…|[Chenyang Lyu, Ch…|
CoRR| 12791182|abs/2010.01785|          null|  7318490|  0|
|        2|           null|Customer-focused …|     [Steven Gordon]|generate
compelli…|     null|            9|           85|  7318490|  1|
|        3|        8969478|Total Variation F…|[Ahmed B. Altamim…|
IEEE Access| 12792132|            8|          null|  7318490|  2|
|        4|        7156412|Enhancing the Stu…|[Antoni Lluís Mes…|   IEEE
Trans. Educ.| 12791148|           64|             1|  7318490|  3|
|        5|           null|Secured zero admi…|       [Dwayne Lane]|facilitate
effici…|     null|          169|           99|  7318490|  4|
|        6|           null|Inverse attitude-…|[Matthew Gill, Ni…|harness
web-enabl…|     null|          224|          123|  7318490|  5|
|        1|        8255791|On the Calculatio…|[R. L. Smelianski…| Inf.
Process. Lett.| 12791629|           22|             5|  8067977|  6|
|        2|        8159350|A high-order fuzz…|       [Mu-Yen Chen]|Future
Gener. Com…| 12791529|           37|          null|  8067977|  7|
|        3|        6553754|Attitude Estimati…|[Byungjin Lee, Ju…|IEEE Trans.
Instr…| 12790705|           70|          null|  8067977|  8|
|        4|           null|Configurable 24/7…|[Joan Lambert, Li…|re-
intermediate l…|     null|          499|          167|  8067977|  9|
|        5|        6849169|Automatic evaluat…|[Akinori Ito, Mas…|Comput.
Speech Lang.| 12790941|           28|             2|  8067977| 10|
|        6|        8070896|Acquiring and App…|[Debbie Richards …|J. Inf.
Knowl. Ma…| 12791460|            2|             2|  8067977| 11|
|        7|        7330047|Stochastic Optimi…|     [Rong Jin 0001]|
CoRR| 12791182| abs/1312.0048|          null|  8067977| 12|
|        8|        9508705|       SIGCOMM news.|    [Erich M. Nahum]|Comput.
Commun. Rev.| 12792585|           36|             5|  8067977| 13|
|        9|        6509187|Stabilizing unsta…|[Ahmet Cetinkaya,…|Syst.
Control. Lett.| 12790686|          113|          null|  8067977| 14|
|        1|           null|Progressive full-…|[Casey Harris, An…|target
distribute…|     null|          312|           39|  7289500| 15|
|        2|           null|Distributed multi…|[Stephen Oconnor,…|incentivize
dot-c…|     null|           58|             7|  7289500| 16|
|        1|        8245635|Investigation of …|[Li Li 0013, Qi W…|J. Intell.
Transp…| 12791625|           18|             3|  9151933| 17|
|        2|        7846928|First British Com…|                null|
Comput. J.| 12791305|            1|             3|  9151933| 18|
|        3|        8521523|Neuro-fuzzy archi…|[Bogdan M. Wilamo…|IEEE Trans.
Ind. …| 12791844|           46|             6|  9151933| 19|
+---------+---------------+--------------------+--------------------+-----------
---------+---------+-------------+-------------+---------+---+
only showing top 20 rows
```

We now print a short overview of the size of all datasets.

Finally, after the import process we save all the dataframes to a *Parquet* file in order to cache the operations we made until now. Saving and reloading the dataframes also solves a known Spark bug related to re-computing UDFs, which would be problematic since our UDF outputs are random. More info here: https://stackoverflow.com/questions/40320563/

[125]:
```
print("Articles", df_articles.count())
print("Authors", df_authors.count())
print("Journals", df_journals.count())
print("References", df_references.count())

df_articles.write.mode('overwrite').parquet('df_articles.parquet')
df_articles_authors.write.mode('overwrite').parquet('df_articles_authors.
 ↪parquet')
df_authors.write.mode('overwrite').parquet('df_authors.parquet')
df_journals.write.mode('overwrite').parquet('df_journals.parquet')
df_references.write.mode('overwrite').parquet('df_references.parquet')
```

```
Articles 10000
Authors 30643
Journals 1955
References 79752
```

0.4 Query

[11]:
```
df_articles = spark.read.parquet('df_articles.parquet')
df_articles_authors = spark.read.parquet('df_articles_authors.parquet')
df_authors = spark.read.parquet('df_authors.parquet')
df_journals = spark.read.parquet('df_journals.parquet')
df_references = spark.read.parquet('df_references.parquet')
```

Actual schemas for the queries

[12]:
```
df_articles.printSchema()
df_articles_authors.printSchema()
df_authors.printSchema()
df_journals.printSchema()
df_references.printSchema()
```

```
root
 |-- ID: integer (nullable = true)
 |-- Title: string (nullable = true)
 |-- MetadataDate: timestamp (nullable = true)
 |-- Volume: string (nullable = true)
 |-- NumberInVolume: string (nullable = true)
 |-- PagesInVolume: string (nullable = true)
 |-- ObjectIds: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- JournalId: integer (nullable = true)

root
 |-- ArticleId: integer (nullable = true)
 |-- AuthorId: integer (nullable = true)

root
 |-- ID: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Email: string (nullable = true)
 |-- Affiliation: string (nullable = true)
 |-- Bio: string (nullable = true)

root
 |-- ID: integer (nullable = true)
 |-- Name: string (nullable = true)

root
 |-- RefNumber: integer (nullable = true)
 |-- InternalPaperId: integer (nullable = true)
 |-- Title: string (nullable = true)
 |-- Authors: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- Journal: string (nullable = true)
 |-- JournalId: integer (nullable = true)
 |-- Volume: string (nullable = true)
 |-- NumberInVolume: string (nullable = true)
 |-- ArticleId: integer (nullable = true)
 |-- ID: long (nullable = true)
```

0.4.1 5 data creation/update queries

01. Insert a new article in the relative DataFrame We extract the article with the max id in order to guarantee that the inserted id is unique because computed as maximum plus one. We compute the execution time in two cases: the first one considers the case in which the id may not be unique and does not insert the article in that case, but the presence of the id in the database is to be checked; the second one just inserts the new article, assuming the id is unique.

[13]:
```
new_article_id = df_articles.groupBy().max("ID").collect()[0][0] + 1
new_article_journal_id = df_journals.filter(df_journals.Name == "World Wide␣
  ↪Web").select("ID").collect()[0][0]
df_new_article = spark.createDataFrame(data = [(new_article_id, "On The Origin␣
  ↪Of Species", None, new_article_journal_id, None, None, None, None)],␣
  ↪schema=df_articles.schema)
```

```
[14]: start_time = time()
      # check if the primary key is already present
      temp_article = df_articles.filter(df_articles.ID == new_article_id)
      if temp_article.isEmpty():
          df_articles = df_articles.union(df_new_article)
          print("Article inserted")
      else:
          print("Article id already present")

      print(f"Execution time: {time() - start_time}")
```

```
Article inserted
Execution time: 0.07419419288635254
```

```
[15]: start_time = time()

      df_articles = df_articles.union(df_new_article)
      df_articles.collect()

      print(f"Execution time: {time() - start_time}")
```

```
[Stage 94:===================>                                      (1 + 1) / 3]

Execution time: 1.206272840499878
```

02. Insert a reference as an article in the database This query consists of three steps:

- the first one is to acquire a single reference which has no internal reference
- the second one is to create the new article DataFrame from the reference data
- the third one is to insert the new article in the DataFrame through the union

```
[16]: start_time = time()

      source_reference = df_references.filter(df_references.InternalPaperId.isNull()).
        ↪limit(1).collect()[0]
      print(source_reference)

      print(f"Execution time: {time() - start_time}")
```

```
Row(RefNumber=1, InternalPaperId=None, Title='Customer-focused hybrid alliance:
Balanced web-enabled instruction set', Authors=['Steven Gordon', 'Sarah Castillo
MD', 'Margaret Saunders', 'Douglas Cummings'], Journal='empower ubiquitous
vortals', JournalId=None, Volume='63', NumberInVolume='86', ArticleId=7318490,
ID=0)
Execution time: 0.11755108833312988
```

[17]:
```
new_article_id = df_articles.groupBy().max("ID").collect()[0][0] + 1

data = [(new_article_id,
    source_reference["Title"],
    datetime.now(),
    source_reference["Volume"],
    source_reference["NumberInVolume"],
    None,
    None,
    source_reference["JournalId"]
)]

df_new_article = spark.createDataFrame(data = data, schema=df_articles.schema)
```

[18]:
```
start_time = time()

df_articles = df_articles.union(df_new_article)
df_articles.collect()

print(f"Execution time: {time() - start_time}")
```

```
Execution time: 0.42409396171569824
```

03. Update the metadata version of the articles of a given journal This query updates the metadata version of all the articles published on a given journal to the current date. In order to do that, the following steps are performed:

- Retrieve a journal name from the database
- Retrieve the articles related to the journal
- Creation of the new DataFrame with the articles' metadata version updated
- Replacement of the old articles with the new ones in the DataFrame (subtract + union)

[19]:
```
journal_name = df_journals.limit(10).collect()[0]["Name"]
journal_name
```

[19]:
```
'World Wide Web'
```

[20]:
```
# find the articles to be updated
start_time = time()

df_articles_to_update = df_articles\
    .join(df_journals, df_articles.JournalId == df_journals.ID)\
    .filter(df_journals.Name == journal_name)\
    .select(
        df_articles.ID,
        df_articles.Title,
        df_articles.MetadataDate,
        df_articles.Volume,
        df_articles.NumberInVolume,
        df_articles.PagesInVolume,
        df_articles.ObjectIds,
        df_articles.JournalId
    )

articles_to_update = df_articles_to_update.collect()

print(f"Execution time: {time() - start_time}")

print(articles_to_update)
```

```
Execution time: 0.4636058807373047
[Row(ID=7154398, Title='Direction-based spatial skyline for retrieving
surrounding objects.', MetadataDate=datetime.datetime(2021, 7, 25, 0, 0),
Volume='23', NumberInVolume='1', PagesInVolume='207-239',
ObjectIds=['https://doi.org/10.1007/s11280-019-00694-w'], JournalId=12790675),
Row(ID=7154450, Title='A threat recognition solution of edge data security in
industrial internet.', MetadataDate=datetime.datetime(2022, 10, 18, 0, 0),
Volume='25', NumberInVolume='5', PagesInVolume='2109-2138',
ObjectIds=['https://doi.org/10.1007/s11280-022-01054-x'], JournalId=12790675),
Row(ID=7153948, Title='Prediction of Web Page Accesses by Proxy Server Log.',
MetadataDate=datetime.datetime(2017, 5, 20, 0, 0), Volume='5',
NumberInVolume='1', PagesInVolume='67-88',
ObjectIds=['https://doi.org/10.1023/A:1015750423727'], JournalId=12790675),
Row(ID=7154603, Title='Enhancing decision-making in user-centered web
development: a methodology for card-sorting analysis.',
MetadataDate=datetime.datetime(2022, 5, 13, 0, 0), Volume='24',
NumberInVolume='6', PagesInVolume='2099-2137',
ObjectIds=['https://doi.org/10.1007/s11280-021-00950-y'], JournalId=12790675)]
```

[21]:
```
# preparation of the data

data: list[tuple[int, str, datetime, str, str, str, list[str], int]] = []
for article in articles_to_update:
    data.append((
        article["ID"],
        article["Title"],
        datetime.now(),
        article["Volume"],
        article["NumberInVolume"],
        article["PagesInVolume"],
        article["ObjectIds"],
        article["JournalId"]
    ))
```

```
df_updated_articles = spark.createDataFrame(data = data, schema=df_articles.
  ↪schema)
```

```
[22]: start_time = time()

# remove the old articles and add the new ones
df_articles = df_articles.subtract(df_articles_to_update)
df_articles = df_articles.union(df_updated_articles)

df_articles.collect()

print(f"Execution time: {time() - start_time}")
```

```
Execution time: 1.5061960220336914
```

04. Change the affiliation of an author We take a random author to change his/her affiliation to USI. First the author is retrieved, then the dataframe of the new author is created with the modified data according to the new affiliation. Working with Spark, in order to change a single value in a column we need to create a new dataframe without the author and then create another dataframe as its union with the dataframe of the modified author. This is because the DataFrame structure of Spark is immutable.

```
[23]: author = df_authors.limit(1).collect()[0]
print("Before: Author", author["Name"], "with affiliation",␣
  ↪author["Affiliation"])

data_new_author: list[tuple[int, str, str, str, str]] = []
data_new_author.append((author["ID"], author["Name"], author["Email"], "USI",␣
  ↪author["Bio"]))
df_new_author = spark.createDataFrame(data=data_new_author, schema=df_authors.
  ↪schema)

# remove the old author and add the new one
df_authors = df_authors.filter((df_authors.ID != author["ID"]))
df_authors = df_authors.union(df_new_author)

author = df_authors.filter(df_authors.ID == author["ID"]).collect()[0]
print("After: Author", author["Name"], "with affiliation",␣
  ↪author["Affiliation"])
```

```
Before: Author Adnan Abid with affiliation University of Voznesensk
After: Author Adnan Abid with affiliation USI
```

05. Remove duplicates in all dataframes Since in theory it is possible to have duplicate rows in a Pyspark DataFrame, it could be useful to check their presence and remove them in case some are found. Anyway, the data creation pipeline we designed ensures that the IDs are unique.

```
[24]: df_articles = df_articles.distinct()
      df_articles_authors = df_articles_authors.distinct()
      df_authors = df_authors.distinct()
      df_journals = df_journals.distinct()
      df_references = df_references.distinct()
```

0.4.2 10 queries with specified complexity

Count the number of internal references to clean Some internal references may be not present as articles in the database: this can happen after some data change, such as the removal of an article from the database. This query retrieves the number of references with are not present in the database.

```
[25]: df_consistent_references = df_references.join(df_articles, df_references.
        ↪InternalPaperId == df_articles.ID)\
          .select("RefNumber",
              "InternalPaperId",
              df_references.Title,
              "Authors",
              "Journal",
              df_references.JournalId,
              df_references.Volume,
              df_references.NumberInVolume,
              "ArticleId",
              df_references.ID
          )

      df_old_references = df_references.subtract(df_consistent_references)

      df_old_references.groupBy().count().show()
```

```
+-----+
|count|
+-----+
|79610|
+-----+
```

01. Retrieve the articles of a journal given the journal's name Complexity: WHERE, JOIN

This query starts from a given journal name and retrieves all the articles published on that journal. The performed steps are the following:

- Retrieve the journal ID given the name
- Join with the articles on the journal ID
- Drop the columns related to the journal

[26]:
```
journal_name = "World Wide Web"

start_time = time()

df_result = df_journals\
    .filter(df_journals.Name == journal_name)\
    .join(df_articles, df_articles.JournalId == df_journals.ID)\
    .drop(df_journals.ID)\
    .drop(df_journals.Name)
df_result.show()

print(f"Execution time: {time() - start_time}")
```

```
+-------+--------------------+--------------------+------+--------------+-------
------+--------------------+---------+
|     ID|               Title|
MetadataDate|Volume|NumberInVolume|PagesInVolume|           ObjectIds|JournalId|
+-------+--------------------+--------------------+------+--------------+-------
------+--------------------+---------+
|7154603|Enhancing decisio…|2022-12-18 17:46:…|    24|             6|
2099-2137|[https://doi.org/…| 12790675|
|7154450|A threat recognit…|2022-12-18 17:46:…|    25|             5|
2109-2138|[https://doi.org/…| 12790675|
|7154398|Direction-based s…|2022-12-18 17:46:…|    23|             1|
207-239|[https://doi.org/…| 12790675|
|7153948|Prediction of Web…|2022-12-18 17:46:…|     5|             1|
67-88|[https://doi.org/…| 12790675|
+-------+--------------------+--------------------+------+--------------+-------
------+--------------------+---------+

Execution time: 1.213135004043579
```

02. Retrieve 5 articles without the DOI registered which have been recently published
Complexity: WHERE, LIMIT, LIKE

This query retrieves 5 articles for which the DOI is registered in the database and which have been published this year. The steps of the query are the following:

- Explode the ObjectIds column of articles
- Filter the articles which have been published this year
- Filter the articles which have an ObjectId containing a DOI reference
- Limit the query to 5 entries

[27]:
```
start_time = time()

df_result = df_articles\
    .select(col("Title"), col("MetadataDate"), explode("ObjectIds"))\
    .withColumnRenamed("col", "ObjectId")\
    .filter(df_articles.MetadataDate > datetime(datetime.now().year, 1, 1))\
    .filter(col("ObjectId").startswith("https://doi.org/"))\
    .limit(5)

df_result.show()

print(f"Execution time: {time() - start_time}")
```

```
+--------------------+-------------------+--------------------+
|               Title|       MetadataDate|            ObjectId|
+--------------------+-------------------+--------------------+
|Sparse Count Data…|2022-03-16 00:00:00|https://doi.org/1…|
|Automatic classif…|2022-11-11 00:00:00|https://doi.org/1…|
|ST-FMT*: A Fast O…|2022-01-08 00:00:00|https://doi.org/1…|
|DOA estimation al…|2022-08-16 00:00:00|https://doi.org/1…|
|Uniform convergen…|2022-06-23 00:00:00|https://doi.org/1…|
+--------------------+-------------------+--------------------+

Execution time: 1.1717472076416016
```

03. Articles of the authors with a given affiliation Complexity: WHERE, IN, Nested Query

This query retrieves all the articles with some author related to a given affiliation. The steps of the query are the following:

- Filter the authors with the given affiliation and create a list
- Filter the bridge table between authors and articles with author IDs in the previous list, and create a list of article IDs out of them
- Filter the articles DataFrame with the article ID in the just created list, and select the name of the articles

[39]:
```
affiliation = "University of Vernon"

start_time = time()

df_authors_of_affiliation = df_authors\
    .filter(df_authors.Affiliation == affiliation)

authors_of_affiliation = [author["ID"] for author in df_authors_of_affiliation.
 ↪collect()]

df_articles_of_authors_of_affiliation = df_articles_authors\
    .join(df_articles, df_articles["ID"] == df_articles_authors["ArticleId"],␣
 ↪'inner') \
    .filter(df_articles_authors.AuthorId.isin(authors_of_affiliation)) \
    .select("ArticleId")
```

```
articles_of_authors_of_affiliation = [article["ArticleId"] for article in␣
↪df_articles_of_authors_of_affiliation.collect()]

df_result = df_articles\
    .filter(df_articles.ID.isin(articles_of_authors_of_affiliation))\
    .select(df_articles.Title)

df_result.show(truncate=False)

print(f"Execution time: {time() - start_time}")
```

```
+--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-----------+
|Title
|
+--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-----------+
|Almost winning: Induced MEG theta power in insula and orbitofrontal cortex
increases during gambling near-misses and is associated with BOLD signal and
gambling severity.|
+--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-----------+

Execution time: 2.1684679985046387
```

04. Count the number of publications on a few journals Complexity: GROUP BY, JOIN, AS

This query counts the publications on each element of a list of journals. The steps of the query are the following:

- Filter the journal DataFrame for journals in the given list
- Join with articles on journal ID
- Group by journal name (journal ID works as well)
- Count the entries for each journal name

[29]:
```
journal_names = ["World Wide Web", "SIGMOD Record"]

start_time = time()

df_result = df_journals\
    .filter(df_journals.Name.isin(journal_names))\
    .join(df_articles, df_journals.ID == df_articles.JournalId, "inner")\
    .groupBy(df_journals.Name)\
    .count()\
    .withColumnRenamed("count", "Publications")

df_result.show()

print(f"Execution time: {time() - start_time}")
```

```
+--------------+------------+
|          Name|Publications|
+--------------+------------+
|World Wide Web|           4|
+--------------+------------+

Execution time: 1.2528290748596191
```

05. Count the registered contacts for each affiliation Complexity: WHERE, GROUP BY

This query counts how many authors for each affiliation have a registered email, from which they can be contacted. The steps of the query are the following:

- Filter the authors who have a registered email
- Group by affiliation
- Count the number of entries for each affiliation

[30]:
```
start_time = time()

df_result = df_authors\
    .filter(df_authors.Email.isNotNull())\
    .groupBy(df_authors.Affiliation)\
    .count()\
    .withColumnRenamed("count", "Number of contacts")

df_result.show()

print(f"Execution time: {time() - start_time}")
```

```
+--------------------+------------------+
|         Affiliation|Number of contacts|
+--------------------+------------------+
|University of Parola|                61|
|University of Ksa…|                63|
|University of Pri…|                83|
|University of Vahdat|               122|
|   University of Ina|               128|
| University of Split|                57|
|University of Bat…|                52|
|University of Gua…|                71|
|University of Nonsan|                73|
|University of Dav…|                73|
|University of Pal…|                58|
|University of Kas…|               129|
|University of Sai…|                54|
|University of Pil…|                73|
|University of Sch…|                51|
|University of Mar…|                58|
|University of Alb…|                66|
|University of Mar…|                58|
|University of Wes…|                59|
|University of Hun…|               172|
+--------------------+------------------+
only showing top 20 rows

Execution time: 0.532426118850708
```

06. Journals with many publications Complexity: GROUP BY, HAVING, AS

This query retrieves the IDs of the journals for which more than 5 publications are registered in the database. The steps are the following:

- Group articles by journal ID
- Count the entries for each group
- Filter the entries with more than 5 publications

[31]:
```
start_time = time()

df_result = df_articles\
    .groupBy(df_articles.JournalId)\
    .agg(count(df_articles.ID).alias("Number of publications"))\
    .filter(col("Number of publications") > 5)

df_result.show()

print(f"Execution time: {time() - start_time}")
```

```
+----------+----------------------+
|JournalId|Number of publications|
+----------+----------------------+
| 12792492|                    21|
| 12791222|                     6|
| 12791288|                    23|
| 12792037|                    24|
| 12791371|                    22|
| 12791712|                     6|
| 12791500|                     6|
| 12791219|                     8|
| 12790733|                    10|
| 12792375|                     9|
| 12790705|                    48|
| 12791668|                    11|
| 12792121|                    6|
| 12790862|                   19|
| 12792462|                   16|
| 12790945|                   45|
| 12790804|                   11|
| 12791047|                   14|
| 12792154|                   19|
| 12791637|                   15|
+---------+---------------------+
only showing top 20 rows

Execution time: 1.3681659698486328
```

07. Registered references with volume cited only once Complexity: WHERE, GROUP BY, HAVING, AS

This query retrieves the citations which refer to an article registered in the database and which are the only ones for a specific volume of a journal. The steps are the following:

- Filter the references which have an internal paper id (they do refer to a registered article)
- Group by journal and volume
- Count the entries for each combination of journal and volume
- Filter the combinations with only one citation

[32]:
```
start_time = time()

df_result = df_references\
    .filter(df_references.InternalPaperId.isNotNull())\
    .groupBy(df_references.Journal, df_references.Volume)\
    .agg(count(df_references.ID).alias("Citations"))\
    .filter(col("Citations") == 1)

df_result.show()

print(f"Execution time: {time() - start_time}")
```

```
+--------------------+--------------+---------+
|             Journal|        Volume|Citations|
+--------------------+--------------+---------+
|       J. Appl. Log.|            15|        1|
|Model. Assist. St…|            14|        1|
|J. Optim. Theory …|           190|        1|
|Multimodal Techno…|             5|        1|
|Math. Comput. Model.|            50|        1|
|          J. Sensors|          2018|        1|
|Comput. Commun. Rev.|            31|        1|
|Discuss. Math. Gr…|            32|        1|
|         SIGACT News|            41|        1|
|  Comput. Chem. Eng.|           121|        1|
|Int. J. Bus. Inf…|           25|        1|
|      Scientometrics|              2|        1|
|J. Inf. Process. …|           20|        1|
|          J. Sensors|           2021|        1|
|           Computing|             64|        1|
|  IET Image Process.|             12|        1|
|            Robotica|             37|        1|
|IEEE Wirel. Commu…|            3|        1|
|                CoRR|abs/1903.09080|        1|
|IEEE Signal Proce…|           22|        1|
+--------------------+--------------+---------+
only showing top 20 rows

Execution time: 0.7499880790710449
```

08. Number of citations for each author of an affiliation Complexity: WHERE, Nested Query, GROUP BY

This query retrieves the number of citations for each author of a given affiliation. The steps are the following:

- Filter authors of the given affiliation and create a list of author names
- Explode authors in references
- Filter references of authors in the list
- Group by author
- Count the number of entries

[33]:
```
affiliation = "University of Gorakhpur"

start_time = time()

df_sub_result = df_authors\
    .filter(col("Affiliation") == affiliation)

sub_result = [author["Name"] for author in df_sub_result.collect()]

df_result = df_references\
    .select(df_references.Title, explode(df_references.Authors).
  ↪alias("Author"))\
    .filter(col("Author").isin(sub_result))\
    .groupBy(col("Author"))\
    .agg(count(df_references.Title).alias("Citations"))

df_result.show()

print(f"Execution time: {time() - start_time}")
```

```
+--------------------+---------+
|              Author|Citations|
+--------------------+---------+
|     Tatiana Glotova|        1|
|    Charles H. Knapp|        6|
|      Jonathan Huang|        3|
|Md. Jahangir Hoss…|        5|
|         Xiaoli Wang|        2|
|Abdon E. Choque R…|        4|
|             Zhen Xu|        4|
|         Qinglai Guo|        2|
|    Hugh E. Williams|        2|
|    Sotirios Brotsis|        4|
|        Ryan Cordell|        5|
|   James Guo Ming Fu|        8|
|   Pavel A. Brodskiy|        4|
|         Shuai Zheng|        7|
|          Danya Song|       10|
|  Alexander Roßnagel|        4|
|      Rajendra Kumar|        6|
|    Stefan Buckreuss|        3|
|           Yupu Yang|        2|
|         Yi Liu 0053|        4|
+--------------------+---------+
only showing top 20 rows

Execution time: 1.0813488960266113
```

09. Titles of internal journals with a reference count in [200, 1000) Complexity: WHERE, GROUP BY, HAVING, 1 JOIN

This query computes the names of the journals that have between 200 and 999 references in the dataset. Results are returned in decreasing order of the reference count. The steps are the following:

- Filter for internal references (i.e. when the journal id is not NULL);
- Group references by internal journal id;
- Count references and filter for the [200,1000) range;
- Perform an INNER JOIN with the journals dataframe;
- Select the name and the count.

[123]:
```python
start_time = time()

result = df_references \
    .filter(F.col("JournalId").isNotNull()) \
    .groupBy("JournalId") \
    .count() \
    .filter((F.col("count") >= 200) & (F.col("count") < 1000)) \
    .join(df_journals, df_journals["Id"] == F.col("JournalId"), 'inner') \
    .select("Name", "count") \
    .sort(F.col("count").desc()) \
    .collect()
```

```
print(f"Execution time: {time() - start_time}")

print(result)
```

```
Execution time: 0.8328709602355957
[Row(Name='IEEE Access', count=848), Row(Name='Sensors', count=643),
Row(Name='Remote. Sens.', count=367), Row(Name='NeuroImage', count=294),
Row(Name='IACR Cryptol. ePrint Arch.', count=288), Row(Name='Neurocomputing',
count=276), Row(Name='Appl. Math. Comput.', count=269), Row(Name='Expert Syst.
Appl.', count=248), Row(Name='Discret. Math.', count=231), Row(Name='IEEE Trans.
Geosci. Remote. Sens.', count=229), Row(Name='Eur. J. Oper. Res.', count=227),
Row(Name='IEEE Trans. Commun.', count=222), Row(Name='IEEE Trans. Signal
Process.', count=211)]
```

10. Top 5 authors by published articles Complexity: WHERE, GROUP BY, HAVING, 2 JOINs

This cell computes the top 5 authors by number of articles published in the last 3000 articles (by ID) in the dataset. We also require the number of articles published to be greater than 1. The steps are the following:

- Filter for the last 3000 rows of the dataset;
- Inner join with the articles-authors dataframe;
- Group by author id;
- Join with the authors dataframe;
- Filter by count greater than 1;
- Select name and count;
- Sort and limit to 5.

[122]:
```
start_time = time()

result = df_articles \
    .filter(F.col("ID") > 7000) \
    .join(df_articles_authors, df_articles_authors["ArticleId"] == F.col("ID"))␣
 ↪\
    .drop("ID") \
    .groupBy("AuthorId") \
    .count() \
    .join(df_authors, df_authors["ID"] == F.col("AuthorId")) \
    .filter(F.col("count") >= 2) \
    .select("Name", "count") \
    .sort(F.col("Count").desc()) \
    .limit(5) \
    .collect()

print(f"Execution time: {time() - start_time}")
```

```python
print(result)
```

```
[Stage 2138:===========================================>             (3 + 1) / 4]

Execution time: 29.39945387840271
[Row(Name='H. Vincent Poor', count=9), Row(Name='Mohamed-Slim Alouini',
count=6), Row(Name='Licheng Jiao', count=6), Row(Name='Kim-Kwang Raymond Choo',
count=6), Row(Name='Sam Kwong', count=6)]
```