



Solution – Exercise 04

Spark, PySpark, Jupyter



Solution

Prerequisites:

- Start Gcloud instance
- Pull and start Docker image (`marcelmittelstaedt/spark_base:latest`)
- Start HDFS and YARN
- Start Jupyter Notebook
- Execute all preparation and example tasks of previous HandsOn slides of last lecture

See:

https://github.com/marcelmittelstaedt/BigData/tree/master/solutions/winter_semester_2023-2024/04_spark_pyspark_jupyter/Solutions.html

...for complete solution (Jupyter Notebook).



Solution

1.) Start Spark Session:

```
# Import Spark Libraries
import findspark, os
findspark.init('/home/hadoop/spark')
from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder \
    .master("yarn") \
    .appName("Jupyter/PySpark Exercises") \
    .enableHiveSupport() \
    .getOrCreate()
```



Solution

2. Create External Spark Table `title_ratings` on HDFS containing data of IMDb file `title.ratings.tsv`

```
# EXERCISE 2) Create External Spark Table title_ratings on HDFS containing data of IMDb file title.ratings.tsv
```

```
# Read IMDb title ratings CSV file from HDFS
```

```
df_title_ratings = spark.read \  
  .format('csv') \  
  .options(header='true', delimiter='\\t', nullValue='null', inferSchema='true') \  
  .load('/user/hadoop/imdb/title_ratings/*.tsv')
```

```
# Save Dataframe back to HDFS (partitioned) as EXTERNAL TABLE and Parquet files
```

```
df_title_ratings.write \  
  .format("parquet") \  
  .mode("overwrite") \  
  .option('path', '/user/hadoop/imdb/title_ratings_table') \  
  .saveAsTable('default.title_ratings')
```

```
# Check Results:
```

```
spark.sql('SELECT * FROM default.title_ratings').show(3)
```

```
+-----+-----+-----+  
| tconst|averageRating|numVotes|  
+-----+-----+-----+  
|tt0000001|          5.7|    1685|  
|tt0000002|          6.0|     208|  
|tt0000003|          6.5|    1425|  
+-----+-----+-----+  
only showing top 3 rows
```



Solution

3. Create External Spark Table `name_basics` on HDFS containing data of IMDb file `name.basics.tsv`

```
# EXERCISE 3) Create External Spark Table name_basics on HDFS containing data of IMDb file name.basics.tsv

# Read IMDb name basics CSV file from HDFS
df_name_basics = spark.read \
    .format('csv') \
    .options(header='true', delimiter='\t', nullValue='null', inferSchema='true') \
    .load('/user/hadoop/imdb/name_basics/*.tsv')

# Save Dataframe back to HDFS (partitioned) as EXTERNAL TABLE and Parquet files
df_name_basics.write \
    .format("parquet") \
    .mode("overwrite") \
    .option('path', '/user/hadoop/imdb/name_basics_table') \
    .saveAsTable('default.name_basics')

# Check Results:
spark.sql('SELECT * FROM default.name_basics').show(3)
```

```
+-----+-----+-----+-----+-----+-----+
| nconst| primaryName| birthYear| deathYear| primaryProfession| knownForTitles|
+-----+-----+-----+-----+-----+-----+
|nm2511361| Shane Vahey| \N| \N| writer, editor, pro...| tt2261585, tt01922...|
|nm2511363| Adolf Seilacher| \N| \N| null| \N|
|nm2511364| Nora Brennan| \N| \N| casting_departmenten...| tt4029524, tt77364...|
+-----+-----+-----+-----+-----+-----+
only showing top 3 rows
```



Solution

4.a) How many **movies** and how many **TV series** are within the IMDB dataset?

```
# EXERCISE 4a) How many movies and how many TV series are within the IMDB dataset?
```

```
# Programmatical Approach
```

```
from pyspark.sql.functions import col
df = spark.table('default.title_basics_partitioned') \
    .where(col('titleType').isin(['movie', 'tvSeries'])) \
    .groupBy('titleType') \
    .count()

df.show(100)
```

```
+-----+-----+
|titleType| count|
+-----+-----+
| tvSeries|202321|
|   movie|569437|
+-----+-----+
```

```
# EXERCISE 4a) How many movies and how many TV series are within the IMDB dataset?
```

```
# SQL Approach
```

```
df = spark.sql('''
    SELECT titleType, count(*)
    FROM default.title_basics_partitioned
    WHERE titleType IN ("movie", "tvSeries")
    GROUP BY titleType
''')

df.show(100)
```

```
+-----+-----+
|titleType|count(1)|
+-----+-----+
| tvSeries|  202321|
|   movie|  569437|
+-----+-----+
```



Solution

4.b) Who is the **youngest** actor/writer/... within the dataset?

```
# EXERCISE 4b) Who is the youngest actor/writer/... within the dataset?
```

```
# Programmatical Approach
```

```
from pyspark.sql.functions import col
df = spark.table('default.name_basics') \
    .where(col('birthYear') != '\\N') \
    .sort(col('birthYear').desc())
```

```
df.show(3)
```

nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
nm0894719	Sarah Vernon	2021	\\N	actress	tt0084987,tt0090499
nm11763191	Win Wilson	2020	\\N	null	\\N
nm12122609	Adam James Sanderson	2020	\\N	actor	tt12668798

only showing top 3 rows

```
# EXERCISE 4b) Who is the youngest actor/writer/... within the dataset?
```

```
# SQL Approach
```

```
df = spark.sql(r"SELECT * FROM default.name_basics WHERE birthYear <> '\\N' ORDER BY birthYear DESC")
```

```
df.show(3)
```

nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
nm0894719	Sarah Vernon	2021	\\N	actress	tt0084987,tt0090499
nm11763191	Win Wilson	2020	\\N	null	\\N
nm12122609	Adam James Sanderson	2020	\\N	actor	tt12668798

only showing top 3 rows



Solution

4.c) Create a list (tconst, original_title, start_year, average_rating, num_votes) of movies which are:

- equal or newer than year 2010
- have an average rating equal or better than 8,1
- have been voted more than 100.000 times

```
# EXERCISE 4c) Create a list (tconst, original_title, start_year, average_rating, num_votes) of movies which are:
# - equal or newer than year 2010
# - have an average rating equal or better than 8,1
# - have been voted more than 100.000 times

# Programmatical Approach
from pyspark.sql.functions import col
df_title_basics = spark.table('default.title_basics_partitioned')
df_title_ratings = spark.table('default.title_ratings')

# JOIN Data Frames
joined_df = df_title_basics.join(df_title_ratings, ['tconst'])

# Filter DF
df = joined_df \
    .where(col('startYear') >= '2010') \
    .where(col('averageRating') > 8.1) \
    .where(col('numVotes') > 100000) \
    .select('tconst', 'originalTitle', 'startYear', 'averageRating', 'numVotes')

# Show Result
df.show(10, False)
```

tconst	originalTitle	startYear	averageRating	numVotes
tt7221388	Cobra Kai	2018	8.6	110286
tt4154756	Avengers: Infinity War	2018	8.4	843065
tt4633694	Spider-Man: Into the Spider-Verse	2018	8.4	380545
tt6763664	The Haunting of Hill House	2018	8.6	183333
tt6966692	Green Book	2018	8.2	384828
tt2380307	Coco	2017	8.4	389537
tt3647998	Taboo	2017	8.4	115867
tt3920596	Big Little Lies	2017	8.5	157469
tt5071412	Ozark	2017	8.4	189152
tt5290382	Mindhunter	2017	8.6	218549

only showing top 10 rows

Solution

4.d) Save result of c) as external Spark Table to HDFS.

```
# EXERCISE 4d) Save result of c) as external Spark Table to HDFS.

# Save Dataframe back to HDFS as external table and Parquet files
df.write \
    .format("parquet") \
    .mode("overwrite") \
    .option('path', '/user/hadoop/imdb/top_movies_table') \
    .saveAsTable('default.top_movies')

# Check Result
spark.sql('SELECT * FROM default.top_movies').show(3)
```

tconst	originalTitle	startYear	averageRating	numVotes
tt4158110	Mr. Robot	2015	8.5	334399
tt4508902	One Punch Man: Wa...	2015	8.8	117086
tt2431438	Sense8	2015	8.3	139787

only showing top 3 rows

Solution

5. Create a Spark Table `name_basics_partitioned`, which:

- contains all columns of table `name_basics`
- is partitioned by column `partition_is_alive`, containing:
 - „alive“ in case actor is still alive
 - „dead“ in case actor is already dead

```
# EXERCISE 5) Create a Spark Table name_basics_partitioned, which:
# - contains all columns of table name_basics
# - is partitioned by column partition_is_alive, containing:
#   - „alive“ in case actor is still alive
#   - „dead“ in case actor is already dead

from pyspark.sql.functions import col, when, lit
df = spark.table('default.name_basics')

# Add column 'partition_is_alive'
df_name_basics = df.withColumn('partition_is_alive',
                               when(col('deathYear') == '\\N', lit('alive')).otherwise(lit('dead'))))

# Save Dataframe back to HDFS (partitioned) as EXTERNAL TABLE and Parquet files
df_name_basics.repartition('partition_is_alive').write \
    .format("parquet") \
    .mode("overwrite") \
    .option('path', '/user/hadoop/imdb/name_basics_partitioned_table') \
    .partitionBy('partition_is_alive') \
    .saveAsTable('default.name_basics_partitioned')

# Check Results:
spark.sql('SELECT * FROM default.name_basics_partitioned WHERE primaryName = "Heath Ledger"]').show(3)
```

nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles	partition_is_alive
nm0005132	Heath Ledger	1979	2008	actor,director,so...	tt0147800,tt04685...	dead



Solution

6. Create a partitioned Spark table `imdb_movies_and_ratings_partitioned`, which:

- contains all columns of the two tables `title_basics_partitioned` and `title_ratings` and
- is partitioned by start year of movie (create and add column `partition_year`).

```
# EXERCISE 6) Create a partitioned Spark table imdb_movies_and_ratings_partitioned, which:  
# - contains all columns of the two tables title_basics_partitioned and title_ratings and  
# - is partitioned by start year of movie (create and add column partition_year).
```

```
# Programmatical Approach
```

```
from pyspark.sql.functions import col  
df_title_basics = spark.table('default.title_basics_partitioned')  
df_title_ratings = spark.table('default.title_ratings')
```

```
# Join DataFrames
```

```
joined_df = df_title_basics.join(df_title_ratings, ['tconst'])
```

```
# Add partition column
```

```
df = joined_df.withColumn('partition_year', col('startYear'))
```

```
# Save DataFrame as external Spark table partitioned by column 'partition_year'
```

```
df.repartition('partition_year').write \  
    .format("parquet") \  
    .mode("overwrite") \  
    .option('path', '/user/hadoop/imdb/imdb_movies_and_ratings_partitioned_table') \  
    .partitionBy('partition_year') \  
    .saveAsTable('default.imdb_movies_and_ratings_partitioned')
```

```
# Check Results:
```

```
spark.sql('SELECT tconst, titleType, primaryTitle, startYear, endYear, partition_year '  
          'FROM default.imdb_movies_and_ratings_partitioned').show(3)
```

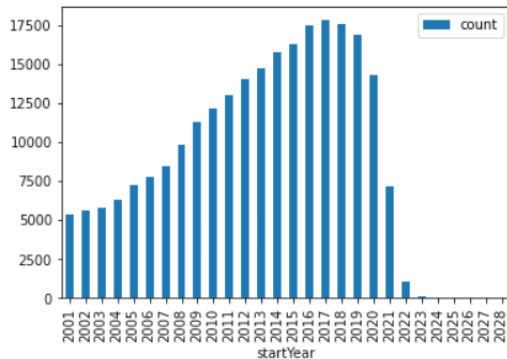
tconst	titleType	primaryTitle	startYear	endYear	partition_year
tt11115836	tvSeries	Slam Dance: The S...	2017	2017	2017
tt11125498	short	Snooze	2017	\N	2017
tt11125898	tvEpisode	Ninovo	2017	\N	2017

only showing top 3 rows



Solution

7. Create following plot, which visualizes:
- the amount of movies (type!)
 - per year
 - since 2000



```
# EXERCISE 7) Create following plot, which visualizes:  
# - the amount of movies (type!)  
# - per year  
# - since 2000  
  
import matplotlib.pyplot as plt  
import pandas  
  
# Create DataFrame to be plotted  
df = spark.table('default.title_basics_partitioned') \\\n    .select('startYear', 'titleType') \\\n    .where(col('startYear') > 2000) \\\n    .where(col('titleType') == 'movie') \\\n    .groupBy('startYear') \\\n    .count() \\\n    .sort(col('startYear').asc())  
  
# Convert Spark DataFrame to Pandas DataFrame  
pandas_df = df.toPandas()  
  
# Plot DataFrame  
pandas_df.plot.bar(x='startYear', y='count')  
  
<AxesSubplot:xlabel='startYear'>
```

