FIT5202 Data processing for Big data¶

Activity: Parallel Aggregation¶

For this tutorial we will implement different operations and aggregations like distinct, group by and order by on Spark DataFrames. In the second part, you will need to use all these operations to answer the lab tasks.

Let's get started.

Table of Contents¶

Import Spark classes and create Spark Context¶

TODO: In the cell block below,

- Create a SparkConfig object with application name set as "Parallel Aggregation"
- specify 2 cores for processing
- Use the configuration object to create a spark session named as **spark**.

Important: You cannot proceed to other steps without completing this.

In [ ]:

```
# TODO: Import libraries needed from pyspark

# TODO: Create Spark Configuration Object

# TODO: Create SparkSession
```

Parallel Aggregation¶

Now we will implement basic aggregation functionalities and visualise the parallelism embedded in Spark as well as the execution plan and functions done to perform these kind of queries.

In this tutorial, you will use two csv files as datasets which contains information about all the athletes that have participated in the Summer and Winter Olympics (athlete_events.csv) as well as the information of their countries (noc_regions.csv).

In [ ]:

```
# Read athlete events data as dataframe
df_events = spark.read.format('csv')\
            .option('header',True).option('escape','"')\
            .load('athlete_events.csv')

# Create Views from Dataframes
df_events.createOrReplaceTempView("sql_events")
```

```
## Verifying the number of partitions for each dataframe
## You can explore the data of each csv file with the function printSchema()
print(f"####### DICTIONARY INFO:")
print(f"Number of partitions: {df_events.rdd.getNumPartitions()}")
df_events.printSchema()
```

Group By ¶

This part contains a simple aggregation query. Look into the query plan and level of parallelism in the Spark UI.

In [ ]:

```
import pyspark.sql.functions as F

#### Aggregate the dataset by 'Year' and count the total number of athletes using
Dataframe
agg_attribute = 'Year'
df_count =
df_events.groupby(agg_attribute).agg(F.count(agg_attribute).alias('Total'))

#### Aggregate the dataset by 'Year' and count the total number of athletes using SQL
sql_count = spark.sql('''
  SELECT year,count(*)
  FROM sql_events
  GROUP BY year
''')
```


In [ ]:

```
df_count.take(5)
```

NOTE: The same thing can be done using `groupby(agg_attribute).agg({'Year':'count'})`

Sort By ¶

We can use orderBy operation to sort the dataframe based on some column.

NOTE: You can specify the sort order using the method **desc()**

`orderBy(df_events.Year.desc())`

In [ ]:

```
df_events.select('Year','Name','Team').orderBy(df_events.Year).show(15)
```

Distinct ¶

This part contains a simple query to get the distinct values of one of the attributes and then sorting them by the same attribute in ascending order.

NOTE: We can use `.sort()` method to do the sorting as well. In the second parameter of the method, we can specify the order of the sorting.

In [ ]:

```
#### Get the distinct values for 'Year' in the dataset using Dataframe
df_distinct_sort = df_events.select('Year').distinct().sort('Year', ascending=True)

#### Get the distinct values for 'Year' in the dataset using SQL
sql_distinct_sort = spark.sql('''
  SELECT distinct Year
  FROM sql_events
  ORDER BY year
''')
df_distinct_sort.take(10)
```

1. Lab Task: Sort the above dataframe i.e. events by **Year** in descending order.

Miscellaneous Dataframe Operations¶

These are the examples of other dataframe operations which are useful.

Describing a Column ¶

The `describe()` melthod gives the statistical summary of the column. If the column is not specified, it gives the summary of the whole dataframe.

In [ ]:
```
df_events.describe('Team').show()
```

Adding and Dropping a column in dataframe ¶

In [ ]:
```
#Here is an example of adding a new column based on the previous column
df_events_new = df_events.withColumn('Years Ago',2020-df_events.Year).select('Years
Ago','Name')
display(df_events_new)
```

TODO: You can use the `.drop('column_name')` method to drop columns from a dataframe. Try this method and drop the column created above.

Using PySpark Functions ¶

You can use PySpark built-in functions along with the `withColumn()` API.

In [ ]:
```
from pyspark.sql import functions as F
from pyspark.sql.types import IntegerType
```



```
#Changing the datatype
#using the display method to see the columns and datatypes of a dataframe
display(df_events)
```


In [ ]:
```
#use CAST to change the datatype of Age Column
df_events = df_events.withColumn('Age',F.col('Age').cast(IntegerType()))
display(df_events)
```


In [ ]:
```
#The folowing example uses another inbuilt function to extract year from the Games
column
df_events = df_events.withColumn('Games Year',F.split(df_events.Games,' ')[0])
df_events.select('Games Year').show(5)
```

DataFrame UDFs (User Defined Functions) ¶

Similar to map operation in an RDDs, sometimes we might want to apply a complex operation to the DataFrame, something which is not provided by the DataFrame APIs. In such scenarios, using a Spark UDF could be handy. To use Spark UDFs, we need to use the F.udf to convert a regular function to a Spark UDF.

In [ ]:
```
#For example, the following function does the same things as the above built-function
but this time we are using a udf
#1. The function is defined
def extract_year(s):
    return int(s.split(' ')[0])

#2. Calling the UDF with DataFrame
from pyspark.sql.functions import udf
from pyspark.sql.types import IntegerType

#First Register the function as UDF
extract_year_udf = udf(extract_year,IntegerType())
```

```
#Call the function
df_events.select('Games',extract_year_udf('Games').alias("Game Year")).show(5)

#4. Calling with Spark SQL
#First Register the function as UDF
spark.udf.register('extract_year',extract_year,IntegerType())

#Call the function
df_events.createOrReplaceTempView('events')
df_sql = spark.sql('''select Games, extract_year(Games) as Game_Year from events''')
df_sql.show(5)
```

Combining DataFrame operations ¶

Now that we have used the main SQL operations to process data, you will implement several queries using Spark Dataframes and SQL to solve each of the queries. The dataset used for this section will be the 2 attached csv files:

- `athlete_events.csv`
- `noc_regions.csv`

The first dataset was already used in the first part of this tutorial. The second one contains the countries with some additional information In this section, you will need to complete most of the code but in some parts, a hint or the name of variables will be given.

In [ ]:


```
# Stop the previous Spark Context to clean all the previous executions from the
previous section
sc.stop()
# Verify that the Spark UI is not running anymore or that there is no content
```


TODO: Since we have removed the Spark Context in the previous code block, start the context once again by using the SparkSession object in the next code block.

In [ ]:


```
# TODO: Import libraries needed from pyspark

# TODO: Create Spark Configuration Object

# TODO: Create SparkSession
```

Create Spark data objects (Dataframes and SQL)¶

In [ ]:

```
# Read athlete events data as dataframe
df_events = spark.read.format('csv')\
            .option('header',True).option('escape','"')\
            .load('data/athlete_events.csv')

# TODO: Read noc regions (countries) data as dataframe
df_regions = spark.read.format('csv')\
            .option('header',True)\
            .load('data/noc_regions.csv')

# Create Views from Dataframes
df_events.createOrReplaceTempView("sql_events")
df_regions.createOrReplaceTempView("sql_regions")

# View Schema for both dataframes
df_events.printSchema()
df_regions.printSchema()
```

Queries/Anaysis¶

For this part, you will need to implement the Dataframe operations and/or the SQL queries to obtain the reports needed for the following questions:

2. Lab Task: Get total number of male athletes per year of the 2000s order by ascending year. **Sample Output:**

```
+----+------------------+
|year|number_of_athletes|
+----+------------------+
|2000|             XXXXX|
|2002|              XXXX|
```

3. Lab Task: Get total number of athletes per Olympic event (summer/winter) in the 1990s decade for Australia and New Zealand. **Sample Output:**

```
+-----------+------+------------------+
|    country|season|number_of_athletes|
+-----------+------+------------------+
|  Australia|Summer|               XXX|
```

TODO: Obtain the minimum, average and maximum height of each country for the Winter Olympics and order by the average value in descending order. **Output should be in the following format:**

```
+--------------------+----------+------------------+----------+
|             country|min_height|        avg_height|max_height|
+--------------------+----------+------------------+----------+
```


TODO: Get the Olympics teams that don't have information of their countries in noc_regions (e.g. Soviet Union since it doesn't exist anymore). **Output should be in the following format:**


```
+--------------------+---+
|                team|noc|
+--------------------+---+
|               Alma|JRS|
|         Australasia|ANZ|
```


Assignment 1

Once you are done with the lab tasks, please work on your Assignment 1.

Congratulations on finishing this activity!

Having practiced today's activities, we're now ready to embark on a trip of the rest of exiciting FIT5202 activities! See you next week!