Notebook

August 10, 2025

0.1 First_steps_Spark

New notebook

1 Working with data in Spark DataFrames

In Spark, an RDD (Resilient Distributed Dataset) is the fundamental data structure. However, in practice, the most commonly used structure is the DataFrame, introduced by Spark SQL. Spark DataFrames are similar to Pandas DataFrames, but they are optimized for large-scale, distributed processing.

1.1 Loading data in a dataframe

We have data in a csv file called *products.csv* in the folder: *Files/data* on a lakehouse.

```
ProductID,ProductName,Category,ListPrice
771,"Mountain-100 Silver, 38",Mountain Bikes,3399.9900
772,"Mountain-100 Silver, 42",Mountain Bikes,3399.9900
773,"Mountain-100 Silver, 44",Mountain Bikes,3399.9900
...
```

Schema Inference Showing first 10 lines in a DataFrame.

```
spark.read.load('path') %Upload a file and transform to df
```

[2]:
```
%%pyspark
df = spark.read.load('Files/products.csv',
    format='csv',
    header = True
)
display(df.limit(10))
```

```
StatementMeta(, 4af69aa4-fe41-42a2-8517-d582a5238835, 4, Finished, Available,␣
↪Finished)
```

```
SynapseWidget(Synapse.DataFrame, e1a6e77b-2b8e-4dc5-88e4-93561e463d4e)
```

`%%pyspark`

In the beginin it calls *magic* and tell that language used in the cell is PySpark.

Specifying an Explicit Schema

You can define an explicit schema for your dataset, which is especially useful when the data file does not include column headers. For example:

```
771,"Mountain-100 Silver, 38",Mountain Bikes,3399.9900
772,"Mountain-100 Silver, 42",Mountain Bikes,3399.9900
773,"Mountain-100 Silver, 44",Mountain Bikes,3399.9900
...
```

[4]:
```
from pyspark.sql.types import *
from pyspark.sql.functions import *

productSchema = StructType([
    StructField("Product_ID", IntegerType()),
    StructField("ProductName", StringType()),
    StructField("Category", StringType()),
    StructField("ListPrice", FloatType()),

])
```

```
StatementMeta(, 4af69aa4-fe41-42a2-8517-d582a5238835, 6, Finished, Available,␣
 ↪Finished)
```

Loading file with no Column and Specifying Explicit Schema

[6]:
```
df = spark.read.load('Files/products_no_header.csv',
    format='csv',
    schema=productSchema,
    header=False
    )

display(df.limit(10))
```

```
StatementMeta(, 4af69aa4-fe41-42a2-8517-d582a5238835, 8, Finished, Available,␣
 ↪Finished)
```

```
SynapseWidget(Synapse.DataFrame, e426d8a8-3faf-42e2-9b0b-0c97cf5d6ba4)
```

Specifying explicit schema improve performance

1.2 Filtering and Grouping of DataFrames

You can use DataFrame's methods to filter, order, group and manage the data:

[10]:
```
pricelist_df = df.select("Product_ID", "ListPrice")
display(pricelist_df.limit(10))
```

```
StatementMeta(, 4af69aa4-fe41-42a2-8517-d582a5238835, 12, Finished, Available,␣
 ↪Finished)
```

```
SynapseWidget(Synapse.DataFrame, 8f07407b-42f8-4b97-9e4a-b7b2bedbd42e)
```

Majority of Methods *select* returns a new object DataFrame. Short sintaxis of *select*:

```
[12]: pricelist_df_short = df["Product_ID", "ListPrice"]
      display(pricelist_df_short.limit(3))
```

```
StatementMeta(, 4af69aa4-fe41-42a2-8517-d582a5238835, 14, Finished, Available,␣
 ↪Finished)
```

```
SynapseWidget(Synapse.DataFrame, 85dfe624-abc1-4e01-ba06-ebd441a4c05e)
```

You can concatenate methods to make a serie of manipulations that generate an **transform DataFrame**

Methods *select* and *where* to create a new DF.

Example

Create a new DF that containsProduct Name and ListPrice where values in Category are "Mountain Bikes" and "Road Bikes"

```
[15]: bikes_df = df.select("ProductName", "Category", "ListPrice").where(
          (df["Category"]=="Mountain Bikes") | (df["Category"]=="Road Bikes")
      )
      display(bikes_df)
```

```
StatementMeta(, 4af69aa4-fe41-42a2-8517-d582a5238835, 17, Finished, Available,␣
 ↪Finished)
```

```
SynapseWidget(Synapse.DataFrame, 04768a4b-d9f2-44b5-8c0a-6613500f0319)
```

Method GroupBy

Counting number of items per category

```
[16]: counts_df = df.select("Product_ID", "Category").groupBy("Category").count()
      display(counts_df)
```

```
StatementMeta(, 4af69aa4-fe41-42a2-8517-d582a5238835, 18, Finished, Available,␣
 ↪Finished)
```

```
SynapseWidget(Synapse.DataFrame, 6c04fad4-3f5a-4be4-80a5-f66ffc9aef8e)
```

1.3 Save a DataFrame

Usually, we want to save processed data with Spark. You can save a DataFrame as a Parquet file in the lakehouse, replacing any existing file with the same name. Parquet is a file format that is highly efficient for large-scale data analysis.

```
[17]: bikes_df.write.mode("overwrite").parquet('Files/bikes.parquet')
```

```
StatementMeta(, 4af69aa4-fe41-42a2-8517-d582a5238835, 19, Finished, Available,␣
 ↪Finished)
```

Creating partitions in output files

Spark optimization methods help maximize performance by efficiently utilizing all job nodes in a cluster.

Example *Saving dataFrame partitioned by* ***Category***

[18]:
```
bikes_df.write.partitionBy("Category").mode("overwrite").parquet("Files/
↪bike_data")
```

```
StatementMeta(, 4af69aa4-fe41-42a2-8517-d582a5238835, 20, Finished, Available,␣
↪Finished)
```

Folder names created include name and value of partition column:

```
Category=Mountain Bikes
Category=Road Bikes
```

Each subfolder contains one or more Parquet files with the product data for the corresponding category.

Note You can partition the data using multiple columns, resulting in a folder hierarchy for each partition key. For example, you could partition sales order data by year and month, so that the folder hierarchy includes one folder for each year value, and within it, a subfolder for each month value.

Load partitionated Data to Notebook

When reading partitioned data into a DataFrame, you can load data from any folder in the hierarchy by specifying explicit values or wildcard characters for the partition fields. In the following example, product data is loaded for the Road Bikes category:

[20]:
```
road_bikes_df = spark.read.parquet('Files/bike_data/Category=Road Bikes')
display(road_bikes_df.limit(3))
```

```
StatementMeta(, 4af69aa4-fe41-42a2-8517-d582a5238835, 22, Finished, Available,␣
↪Finished)
```

```
SynapseWidget(Synapse.DataFrame, 4cd67c21-3cbb-4dd6-829c-182b3413d05b)
```