DATA AND AI

Advanced PySpark Functions

Shwetank Singh
GritSetGrow - GSGLearn.com

gsglearn.com

Data and AI

Advanced PySpark Functions

mapPartitions

Process data in chunks (partitions) instead of row by row to improve performance.

```python
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("MapPartitionsExample").getOrCreate()

# Define a function to process each partition
def process_partition(partition):
    for record in partition:
        record['value'] = record['value'].upper()  # Convert 'value' to uppercase
        yield record

# Create a DataFrame
df = spark.createDataFrame([
    {"id": 1, "value": "hello"},
    {"id": 2, "value": "world"},
    {"id": 3, "value": "pyspark"}
])

processed_df = df.rdd.mapPartitions(process_partition).toDF()

processed_df.show()
```

mapPartitions processes each group of rows (partition) together. In this example, it converts the value field of each record to uppercase within each partition, which can be more efficient than processing each row individually.

Shwetank Singh
GritSetGrow - GSGLearn.com

Data and AI

Advanced PySpark Functions

posexplode

Break down arrays into rows while keeping track of each element's position.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import posexplode

spark = SparkSession.builder.appName("PosexplodeExample").getOrCreate()

df = spark.createDataFrame([
    (1, ["apple", "banana", "cherry"]),
    (2, ["date", "elderberry"])
], ["id", "fruits"])

exploded_df = df.select("id", posexplode("fruits").alias("position", "fruit"))

exploded_df.show()
```

posexplode takes the fruits array and creates a new row for each fruit, also adding a position column that shows the index of each fruit in the array.

Shwetank Singh
GritSetGrow - GSGLearn.com

Data and AI

Advanced PySpark Functions

aggregate (Higher-Order Function)

Perform custom calculations on array or map columns.

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import aggregate, col, lit

spark = SparkSession.builder.appName("AggregateExample").getOrCreate()

df = spark.createDataFrame([
    (1, [2, 3, 4]),
    (2, [5, 6, 7])
], ["id", "numbers"])

aggregated_df = df.withColumn(
    "product",
    aggregate(col("numbers"), lit(1), lambda acc, x: acc * x)
)

aggregated_df.show()
```

aggregate multiplies all numbers in the numbers array for each row, resulting in a new product column that holds the product of the array elements.

Shwetank Singh
GritSetGrow - GSGLearn.com

Data and AI

Advanced PySpark Functions

approxQuantile

Quickly estimate quantiles (like median) for large datasets without needing exact values.

```
df = spark.read.parquet("hdfs:///data/large_dataset.parquet")

quantiles = df.approxQuantile("salary", [0.25, 0.5, 0.75], 0.01)

print(f"Approximate 25th, 50th, 75th percentiles: {quantiles}")
```

approxQuantile provides an estimated value for percentiles, which is much faster and uses less memory than calculating exact quantiles, especially on big data.

Shwetank Singh
GritSetGrow - GSGLearn.com

Data and AI

Advanced PySpark Functions

crosstab

Create a table showing the frequency of combinations between two categorical columns.

```python
from pyspark.sql.functions import crosstab

df = spark.createDataFrame([
    ("Male", "Yes"),
    ("Female", "No"),
    ("Male", "No"),
    ("Female", "Yes"),
    ("Male", "Yes")
], ["gender", "response"])

crosstab_df = df.crosstab("gender", "response")

crosstab_df.show()
```

crosstab generates a matrix that displays how often each pair of categories from two columns occurs, useful for understanding relationships between categories.

Shwetank Singh
GritSetGrow - GSGLearn.com

Data and AI

Advanced PySpark Functions

window with Custom Specifications

Perform calculations across a sliding window of rows, such as running totals or rankings.

```
from pyspark.sql.window import Window
from pyspark.sql.functions import sum, col

df = spark.createDataFrame([
    ("A", "2025-01-01", 100),
    ("A", "2025-01-02", 200),
    ("A", "2025-01-03", 300),
    ("B", "2025-01-01", 400),
    ("B", "2025-01-02", 500)
], ["category", "date", "value"])

window_spec = Window.partitionBy("category") \
                    .orderBy("date") \
                    .rowsBetween(-1, 0)

df_with_running_total = df.withColumn(
    "running_total",
    sum(col("value")).over(window_spec)
)

df_with_running_total.show()
```

Using window functions, you can calculate values like moving averages or ranks within specific groups of data, based on custom rules for how the window is defined.

Shwetank Singh
GritSetGrow - GSGLearn.com

Data and AI

Advanced PySpark Functions

bucketizer

Convert continuous numerical data into discrete buckets or categories.

```python
from pyspark.ml.feature import Bucketizer

df = spark.createDataFrame([
    (0.1,),
    (0.5,),
    (1.0,),
    (1.5,),
    (2.0,),
    (2.5,)
], ["feature"])

splits = [0.0, 1.0, 2.0, 3.0]

bucketizer = Bucketizer(splits=splits, inputCol="feature", outputCol="bucket")
bucketed_df = bucketizer.transform(df)
bucketed_df.show()
```

bucketizer splits numerical values into different ranges (buckets), turning continuous data into categorical data, which is helpful for certain types of analysis or models.

Shwetank Singh
GritSetGrow - GSGLearn.com

Data and AI

Advanced PySpark Functions

pivot

Transform data from a long format to a wide format by turning unique values from one column into multiple columns.

```python
from pyspark.sql.functions import sum, col

df = spark.createDataFrame([
    ("A", "2025-01", 100),
    ("A", "2025-02", 150),
    ("B", "2025-01", 200),
    ("B", "2025-02", 250)
], ["category", "month", "sales"])

pivot_df = df.groupBy("category") \
             .pivot("month") \
             .sum("sales")

pivot_df.show()
```

pivot reshapes your DataFrame by creating new columns based on unique values in the month column, making it easier to compare sales across different months for each category.

Shwetank Singh
GritSetGrow - GSGLearn.com

Data and AI

Advanced PySpark Functions

spark.udf.register

Create and use custom functions that are not available in PySpark's built-in functions.

```python
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

def reverse_string(s):
    return s[::-1]

spark.udf.register("reverse_string", reverse_string, StringType())

df = spark.createDataFrame([
    ("hello",),
    ("world",)
], ["word"])

df.withColumn("reversed", reverse_string(df.word)).show()
```

spark.udf.register lets you define your own function (reverse_string) to reverse strings and apply it to a DataFrame column, extending PySpark's capabilities.

Shwetank Singh
GritSetGrow - GSGLearn.com

Data and AI

Advanced PySpark Functions

broadcast

Optimize join operations by sending a small DataFrame to all worker nodes to avoid data shuffling.

```
from pyspark.sql.functions import broadcast

small_df = spark.createDataFrame([
    (1, "A"),
    (2, "B")
], ["id", "category"])

large_df = spark.createDataFrame([
    (1, "data1"),
    (2, "data2"),
    (3, "data3")
], ["id", "value"])

joined_df = large_df.join(broadcast(small_df), "id")

joined_df.show()
```

broadcast makes the small_df available on all nodes, speeding up the join with large_df by avoiding the shuffle of small_df across the cluster.

Shwetank Singh
GritSetGrow - GSGLearn.com

Data and AI

Advanced PySpark Functions

explode_outer

Expand array or map columns into multiple rows, including rows with empty or null arrays.

```python
from pyspark.sql.functions import explode_outer

df = spark.createDataFrame([
    (1, ["a", "b"]),
    (2, []),
    (3, None)
], ["id", "letters"])

exploded_df = df.select("id", explode_outer("letters").alias("letter"))

exploded_df.show()
```

explode_outer expands the letters array into separate rows but keeps rows with empty or null arrays by assigning null to the letter column, ensuring no data is lost.

Shwetank Singh
GritSetGrow - GSGLearn.com

thank you!

Shwetank Singh
GritSetGrow - GSGLearn.com