09 Sorting -KirkYagami 👨‍💻🕵️

Sorting is a fundamental operation in data processing, enabling the organization of data in a specific order based on one or more columns. PySpark provides robust tools for sorting data, whether working directly with DataFrames or using SQL queries. This lecture covers various aspects of sorting in PySpark, with practical examples using the `disney_raw` dataset.

1. Introduction to Sorting in PySpark

- Sorting arranges the rows in a DataFrame in ascending or descending order based on the values in specified columns.
- It's useful for tasks such as ranking, ordering by relevance, or preparing data for subsequent analysis.

2. Sorting DataFrames

PySpark DataFrames provide two primary methods for sorting:

- `sort()`: Sorts the DataFrame by the specified column(s). Equivalent to SQL `ORDER BY`.
- `orderBy()`: An alias for `sort()`, commonly used and more explicit.

Both methods accept one or more columns, and optionally, the sorting order (ascending or descending).

3. Basic Sorting

- **Ascending Order (Default)**
  By default, PySpark sorts in ascending order.

  ```
  disney_raw.orderBy("year").show(10, truncate=False)
  ```

  This example sorts the `disney_raw` DataFrame by the `year` column in ascending order and displays the first 10 rows.

- **Descending Order**
  To sort in descending order, use the `desc` function from `pyspark.sql.functions`.

  ```
  from pyspark.sql.functions import desc

  disney_raw.orderBy(desc("year")).show(10, truncate=False)
  ```

  This code sorts the DataFrame by the `year` column in descending order.

4. Sorting by Multiple Columns

You can sort by multiple columns by passing them as arguments to the `orderBy()` or `sort()` methods.

- **Example:**

  ```
  disney_raw.orderBy("genre", desc("year")).show(10, truncate=False)
  ```

This sorts the DataFrame first by `genre` in ascending order and then by `year` in descending order within each genre.

5. Sorting and Handling Null Values

By default, null values appear first when sorting in ascending order and last when sorting in descending order. You can control this behavior with the `na.last()` or `na.first()` options:

- **Example:**

```
disney_raw.orderBy(disney_raw["genre"].asc_nulls_last()).show(10, truncate=False)
```

This sorts the `genre` column in ascending order, placing nulls at the end.

- **Excluding Null Values:**

To exclude null values from the sorting operation, you can filter them out before sorting:

```
disney_raw.filter(disney_raw["genre"].isNotNull()) \
    .orderBy("genre").show(10, truncate=False)
```

This filters out rows where `genre` is null before sorting.

6. Sorting in PySpark SQL

If you prefer using SQL queries, PySpark allows you to run SQL-like queries directly on DataFrames.

- **Registering the DataFrame as a Temporary View:**

```
disney_raw.createOrReplaceTempView("disney_view")
```

- **SQL Query for Sorting:**

```
ss.sql("""
    SELECT genre, year, title
    FROM disney_view
    ORDER BY year DESC, title ASC
""").show(10, truncate=False)
```

This query sorts the data by `year` in descending order and by `title` in ascending order within the same year.

7. Performance Considerations

- **Sorting Large DataFrames:**
  Sorting large datasets can be resource-intensive. PySpark's distributed architecture helps by dividing the workload across multiple nodes, but it's essential to optimize your environment:
  - Ensure enough memory is allocated to executors.
  - Leverage sorting operations judiciously in your ETL pipelines.

- **Optimizing with Partitions:**
  When working with large datasets, consider partitioning your data to optimize sorting performance. Sorting within partitions can be faster and reduce shuffle operations.

- **Example:**

```
disney_raw.repartition("genre").orderBy("genre", "year").show(10, truncate=False)
```

  This repartitions the data by `genre` before sorting, which can be more efficient.

8. Real-World Example Using `disney_raw` Dataset

- **Task:** Identify the top 10 genres with the most titles and display them in descending order of count.

```
from pyspark.sql.functions import desc, col

disney_raw.filter(col("genre").isNotNull()) \
    .groupBy("genre") \
    .count() \
    .orderBy(desc("count")) \
    .show(10, truncate=False)
```

  This code filters out null genres, groups by `genre`, counts the number of occurrences, and sorts the results in descending order by count.

9. Conclusion

- Sorting is a powerful tool for organizing and analyzing data. Whether using PySpark DataFrame methods or SQL-like syntax, understanding the nuances of sorting can greatly enhance your data processing capabilities.
- Always consider null handling, performance implications, and specific requirements like multi-column sorting to make the most of sorting operations in PySpark.

This concludes the lecture notes on sorting in PySpark SQL and DataFrames, using practical examples from the `disney_raw` dataset.