Seekho Bigdata
Data is the New Oil

Window Functions

Karthik Kondpak
9989454737

Day 18 — Spark Optimization Topic

🔥 Window Functions — The Core of Analytical Workloads

Window functions allow you to perform **calculations across groups of rows** without collapsing the rows into a single output like `GROUP BY` does.

They are the backbone of:

- Analytics
- Ranking
- Time-series pipelines
- Sessionization
- Customer 360 views
- Fraud detection
- Delta Live Tables transformations

Why Window Functions Matter

✓Let you look at "neighboring rows"

✓Perform calculations without losing row-level detail

Avoid complex self-joins

✓Extremely efficient when used with partitions

✓Used everywhere in Data Engineering + Analytics

✓

Core Window Function Categories

Type	Purpose	Examples
Ranking	Compare row positions	`row_number`, `rank`, `den se_ ra nk`
Analytics	Running totals, moving averages	`lag, lead, sumavg`
Value Functions	Access previous/next rows	`lag, lead`
Distribution	Percentiles	`percent_rank, ntile`

Window Specification

```
from pyspark.sql.window import Window
from pyspark.sql.functions import *

window_spec =
Window.partitionBy("customer_id").orderBy("order_date")
```

A window is defined by:

- PARTITION BY → group
- ORDER BY → sequence
- ROWS BETWEEN → frame (optional)

1. Ranking Functions

row_number()

Assigns unique number within each partition.

```
df = orders.withColumn(
    "rn",
    row_number().over(window_spec)
)
```

```
)
```

Use case: Latest order per customer.

rank()

Gives same rank for ties, but skips numbers.

```
df = orders.withColumn(
    "rank",
    rank().over(window_spec)
)
```

dense_rank()

Same as rank but **no gaps**.

```
dense_rank().over(window_spec)
```

2. Analytics Functions (Running Calculations)

Running total per customer

```
df = orders.withColumn(
    "running_total",
    sum("amount").over(window_spec)
)
```

Moving average (last 3 rows)

```
df=orders.withColumn(
    "moving_avg",
    avg("amount").over(
        window_spec.rowsBetween(-2, 0)
    )
)
```

3. Value Functions (lag/lead)

lag() — previous row

```
df = orders.withColumn(
    "prev_amount",
    lag("amount", 1).over(window_spec)
)
```

lead() — next row

```
lead("amount", 1).over(window_spec)
```

Scenario — Zomato Time-Series

orders:

order_id	customer_id	amount	order_date
101	1	300	2024-11-01
102	1	200	2024-11-05
103	1	400	2024-11-08

Goal: Find days-gap between customer orders

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

result = orders.withColumn(
    "prev_date",
    lag("order_date",
1).over(Window.partitionBy("customer_id").orderBy
("order_date"))
).withColumn(
    "gap_days",
    datediff(col("order_date"), col("prev_date"))
)
```

Advanced Topic — ROWS vs RANGE

ROWS BETWEEN

Count-based frame

→ Last 3 rows

→ Next row

→ Current + Previous

RANGE BETWEEN

Value-based frame

→ All rows with amount <= current row amount

→ All rows within 3 days range

How Window Functions Improve Performance

- Avoids expensive self-joins
- Pushes work to Tungsten engine
- Uses partitioned execution
- Very memory efficient compared to groupBy + join combinations

Let's build your Data Engineering journey together!

Call us directly at: 9989454737

https://seekhobigdata.com/