Seekho Bigdata
Data is the New Oil

PySpark Scenario-Based Interview Questions

DAY 3 — Window Functions

Karthik Kondpak
9989454737

PySpark Scenario-Based Interview Questions

DAY 3 — Window Functions

Window Functions Covered Today

- ROW_NUMBER
- RANK vs DENSE_RANK
- LAG / LEAD
- Running Totals
- Identifying Gaps in Dates

Common Sample Data: `transactions_df`

customer_id	txn_date	amount
101	2024-01-01	1000
101	2024-01-05	1500
101	2024-01-10	2000
102	2024-01-03	3000
102	2024-01-08	2500

✅ Question 1: Assign Transaction Sequence per Customer (ROW_NUMBER)

🔹 Scenario

A fintech company wants to assign a **transaction sequence number** for each customer based on transaction date.

🧪 PySpark Solution

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

window_spec =
Window.partitionBy("customer_id").orderBy("txn_date")

result_df = transactions_df.withColumn(
    "txn_sequence",
    row_number().over(window_spec)
)

result_df.show()
```

📝 Explanation

- `partitionBy()` restarts numbering per customer
- `orderBy()` defines row sequence
- `row_number()` assigns unique sequence numbers

✅ Question 2: Rank Customers by Transaction Amount (RANK vs DENSE_RANK)

🔹 Scenario

An analytics team wants to rank transactions **by amount** per customer.

🧪 PySpark Solution

```
from pyspark.sql.functions import rank, dense_rank

rank_window =
Window. partit ionBy ("cus tomer_ id").or derBy( col(" amoun
").desc())
```

```
ranked_df = transactions_df \
    .withColumn("rank", rank().over(rank_window)) \
    .withColumn("dense_rank",
dense_rank().over(rank_window))

ranked_df.show()
```

📝 **Explanation**

- rank() leaves gaps when values are equal
- dense_rank() does not leave gaps
- Frequently asked **theory + coding** interview question

✅ Question 3: Fetch Previous Transaction Amount (LAG)

🔹 Scenario

A bank wants to compare each transaction with the **previous transaction amount** per customer.

🧪 PySpark Solution

```
from pyspark.sql.functions import lag

lag_df = transactions_df.withColumn(
    "prev_amount",
    lag("amount", 1).over(window_spec)
)

lag_df.show()
```

📝 Explanation

- lag(column, offset) fetches previous row value
- Commonly used in **trend analysis**

✅ Question 4: Fetch Next Transaction Amount (LEAD)

🔹 Scenario

Operations team wants to analyze the **next transaction** value for prediction models.

🧪 PySpark Solution

```
from pyspark.sql.functions import lead

lead_df = transactions_df.withColumn(
    "next_amount",
    lead("amount", 1).over(window_spec)
)

lead_df.show()
```

📝 Explanation

- lead() fetches future row values
- Useful in **forecasting and churn analysis**

✅ Question 5: Calculate Running Total per Customer

🔹 Scenario

A payments company wants to calculate **cumulative spend per customer** over time.

🧪 PySpark Solution

```
from pyspark.sql.functions import sum

running_window = Window.partitionBy("customer_id") \
    .orderBy("txn_date") \
    .rowsBetween(Window.unboundedPreceding,
Window.currentRow)

running_df = transactions_df.withColumn(
    "running_total",
    sum("amount").over(running_window)
)

running_df.show()
```

📝 Explanation

- unboundedPreceding starts from first row
- Used heavily in **financial reporting**

✅ Question 6: Identify Missing Transaction Dates (GAPS)

🔹 Scenario

The data quality team wants to identify **missing transaction dates per customer**.

🧪 PySpark Solution

```
from pyspark.sql.functions import datediff

lag_date_df = transactions_df.withColumn(
    "prev_date",
    lag("txn_date", 1).over(window_spec)
)

gaps_df =
lag_date_df.filter(datediff(col("txn_date"),
col("prev_date")) > 1)

gaps_df.show()
```

📝 **Explanation**

- Compare current and previous dates
- datediff() helps detect missing days
- Frequently asked **real-world data quality question**

Seekho Bigdata

Data is the New Oil

Let’s build your Data Engineering journey together!

📩 Call us directly at: 9989454737

🌐 https://seekhobigdata.com/