Deloitte.

DATA ENGINEER

SCENARIO-BASED INTERVIEW QUESTIONS

AFRIN AHAMED

Question:

- **You are given a stream of IoT sensor data with columns: sensor_id, timestamp, and value. Detect sensors with values exceeding a threshold (e.g., 100) in real-time.**

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

# Initialize Spark session
spark = SparkSession.builder.appName("IoTStreaming").getOrCreate()

# Read streaming data
stream_df = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "localhost:9092").load()

# Parse the data
parsed_df = stream_df.selectExpr("CAST(value AS STRING)").selectExpr("split(value, ',') as data") \
    .selectExpr("CAST(data[0] AS INT) as sensor_id", "CAST(data[1] AS TIMESTAMP) as timestamp", "CAST(data[2] AS FLOAT) as value")

# Filter sensors with values exceeding the threshold
result_df = parsed_df.filter(col("value") > 100)

# Write the output to the console
query = result_df.writeStream.format("console").start()
query.awaitTermination()
```

Explanation:

- Read streaming data from Kafka.
- Parse the data and filter sensors with values exceeding the threshold.
- Write the output to the console.

Question:

- **You are given a stream of log data with columns: timestamp, user_id, and action. Calculate the count of actions per user in real-time.**

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import window

# Initialize Spark session
spark = SparkSession.builder.appName("StreamingAggregation").getOrCreate()

# Read streaming data
stream_df = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "localhost:909
2").load()

# Parse the data
parsed_df = stream_df.selectExpr("CAST(value AS STRING)").selectExpr("split(value, ',') as dat
a") \
    .selectExpr("CAST(data[0] AS TIMESTAMP) as timestamp", "CAST(data[1] AS INT) as user_id", "C
AST(data[2] AS STRING) as action")

# Aggregate actions per user
result_df = parsed_df.groupBy("user_id", window("timestamp", "1 minute")).count()

# Write the output to the console
query = result_df.writeStream.format("console").start()
query.awaitTermination()
```

Explanation:

- Explanation:
- Read streaming data from Kafka.
- Parse the data and group by **user_id** and a 1-minute window.
- Write the output to the console.

Common Mistakes:

- Not defining the window correctly.
- Forgetting to start the streaming query with **awaitTermination()**.

Question:

- **You are given a dataset of user transactions with columns: user_id, transaction_id, and amount. The dataset is heavily skewed on the user_id column. Calculate the total transaction amount per user while handling the skew.**

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

# Initialize Spark session
spark = SparkSession.builder.appName("SkewedAggregation").getOrCreate()

# Load data
transactions_df = spark.read.csv("transactions.csv", header=True, inferSchema=True)

# Add a random salt to the skewed key (user_id)
transactions_df = transactions_df.withColumn("salted_user_id", col("user_id") % 10)

# Aggregate total amount per salted user_id
result_df = transactions_df.groupBy("salted_user_id", "user_id").agg(sum("amount").alias("total_amount"))

# Remove the salt and aggregate again
final_result_df = result_df.groupBy("user_id").agg(sum("total_amount").alias("total_amount"))

# Show result
final_result_df.show()
```

Explanation:

- Add a random salt to the skewed key **(user_id)** to distribute the data evenly.
- Perform the first aggregation on the salted key.
- Remove the salt and perform a second aggregation to get the final result.

Common Mistakes:

- Not addressing data skew, leading to slow performance.
- Forgetting to remove the salt in the final aggregation.

Question:

- **You are given a nested JSON dataset with the following structure:**

```json
{
  "order_id": 1,
  "customer_id": 101,
  "items": [
    {"product_id": 201, "quantity": 2, "price": 10.0},
    {"product_id": 202, "quantity": 1, "price": 20.0}
  ],
  "payment": {
    "method": "credit_card",
    "amount": 30.0
  }
}
```

Write a Spark job to extract the following:

- **Total revenue per order.**
- **Payment method for each order.**

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, explode

# Initialize Spark session
spark = SparkSession.builder.appName("ComplexJSON").getOrCreate()

# Load JSON data
df = spark.read.json("orders.json")

# Explode the nested "items" array
flattened_df = df.withColumn("item", explode("items"))

# Calculate revenue for each item
revenue_df = flattened_df.withColumn("revenue", col("item.quantity") * col("item.price"))

# Aggregate total revenue per order
result_df = revenue_df.groupBy("order_id", "payment.method").agg(sum("revenue").alias("total_revenue"))

# Show result
result_df.show()
```

Explanation:

- Use explode to flatten the nested items array.
- Calculate revenue for each item by multiplying quantity and price.
- Group by order_id and payment.method to aggregate the total revenue.

Common Mistakes:

- Not using explode to handle nested arrays.
- Forgot to include the payment.method in the groupBy.

Think your skills are enough?

Think again—these Data engineer scenario-based questions could cost you your data engineering job.

In a recent interview at many big MNC's, one of our students faced scenario-based questions related to data engineering, and many candidates struggled to answer them correctly. These questions are designed to test your real-world knowledge and ability to solve complex data engineering problems.

Unfortunately, many students failed to answer these questions confidently. The truth is, preparation is key, and that's where Prominent Academy comes in!
We specialize in preparing you for spark and data engineering interviews by:
✅ Offering scenario-based mock interviews
✅ Providing hands-on training with data engineering features
✅ Optimizing your resume & LinkedIn profile
✅ Giving personalized interview coaching to ensure you're job-ready
Don't leave your future to chance!