Datametica Data Engineer Interview Guide – Experienced 3+

Technical - Round 1 and 2 combined

1. Fact and Dimension Tables – Explain with Examples

- **Fact Table**:

  Definition: A fact table stores quantitative data for analysis, such as sales, revenue, or counts. It typically contains keys that reference dimension tables and factual data (measures).

  Example:

  - **Fact Table (Sales)**:

Order_ID	Product_ID	Customer_ID	Sales_Amount	Date
1	101	5001	200	2025-01-01
2	102	5002	150	2025-01-02

- **Dimension Table**:

  Definition: A dimension table provides descriptive attributes related to the business process. It helps in filtering or grouping data.

  Example:

  - **Dimension Table (Product)**:

Product_ID	Product_Name	Category
101	Laptop	Electronics
102	Phone	Electronics

2. Types of Keys – Discuss Primary, Foreign, and Composite Keys

- **Primary Key**:

  Definition: A primary key uniquely identifies each record in a table. It cannot have NULL values.

  Example: In a Customer table, Customer_ID can be a primary key.

- **Foreign Key**:

  Definition: A foreign key is a column that links a record in one table to a record in another table. It points to the primary key of another table.

  Example: In the Order table, Customer_ID could be a foreign key referencing Customer_ID in the Customer table.

- **Composite Key**:

  Definition: A composite key consists of two or more columns that together uniquely identify a record in a table.

  Example: In a Sales table, a combination of Order_ID and Product_ID could serve as a composite key.

3. Spark Architecture – Explain Driver, Executors, and Tasks

- **Driver**:

  The Driver program is responsible for managing the Spark application. It coordinates tasks, schedules jobs, and controls the execution.

  The Driver sends tasks to the Executors for execution.

- **Executors**:

  Executors are worker nodes that perform computations and store data for a Spark application.

  Each executor runs in its own JVM and is responsible for executing a subset of tasks.

- **Tasks**:

  Tasks are the smallest units of work in Spark. They are executed by the Executors, and each task corresponds to a partition of data.

4. Drop Null Values – Example in PySpark

To remove rows with NULL values in PySpark, you can use the .dropna() function:

```
df = df.dropna()
```

- This will remove all rows containing NULL values. If you want to drop rows based on a specific column, you can specify that column:

```
df = df.dropna(subset=["column_name"])
```

5. Transformations in Code – Discuss Common Transformations Used

- **map()**: Applies a function to each element in the RDD.
- **filter()**: Returns a new RDD with elements that satisfy a given condition.
- **flatMap()**: Similar to map(), but it can return zero or more output items for each input item.
- **groupByKey()**: Groups data by the key.
- **reduceByKey()**: Combines values of the same key using a function.
- **join()**: Joins two RDDs based on a key.

Example (filter transformation):

```
rdd = rdd.filter(lambda x: x > 10)
```

6. GroupByKey vs ReduceByKey – Differences and Performance Implications

- **groupByKey()**:

  Definition: Groups data by the key. It can lead to large data shuffling, which is inefficient.

  Performance: Inefficient for large datasets as it requires moving data between nodes for grouping.

- **reduceByKey()**:

  Definition: Combines values with the same key using a function before shuffling, reducing the amount of data moved between nodes.

  Performance: More efficient than groupByKey() because it reduces data before shuffling.

7. Repartition vs Coalesce – Use Cases for Each

- **Repartition**:

  Definition: Increases or decreases the number of partitions in a DataFrame.

  Use Case: When you need to increase the number of partitions for better parallelism in operations like join.

  Example: df.repartition(10)

- **Coalesce**:

  Definition: Reduces the number of partitions by merging them. More efficient than repartition() when reducing the number of partitions.

  Use Case: Used before writing data to disk to minimize file size and avoid small file problems.

  Example: df.coalesce(1)

8. Spark Optimization Techniques – Share Strategies to Improve Performance

- **Avoid Shuffling**: Minimize operations that cause shuffling, such as groupByKey().
- **Partitioning**: Repartition data based on the keys to ensure better parallelism.
- **Broadcast Joins**: Use broadcast joins when one table is much smaller than the other.
- **Caching**: Cache intermediate data for reuse to avoid recomputation.

df.cache()

- **Avoid Skewed Data**: Use salting techniques or custom partitioning when dealing with skewed data.

9. Fill Null Values – Example in PySpark

To fill NULL values in a DataFrame, use .fillna():

```
df = df.fillna({'column_name': 'default_value'})
```

- This fills NULL values in a specific column with a default value. To fill all columns:

```
df = df.fillna('default_value')
```

10. Remove Duplicates – How to Remove Duplicates in PySpark

To remove duplicates from a DataFrame, use .dropDuplicates():

```
df = df.dropDuplicates()
```

- You can also remove duplicates based on specific columns:

```
df = df.dropDuplicates(["column_name"])
```

11. Optimized Join of Large and Small Tables in Spark

- **Broadcast Join**: For joining a large table with a small one, you can use **broadcast joins** to avoid shuffling.

```
from pyspark.sql.functions import broadcast
df_large.join(broadcast(df_small), "key")
```

- **Broadcast** the smaller dataset to all nodes, which reduces the amount of data shuffling and speeds up the join operation.

12. Job/Stage/Task Creation – Explain Spark's Execution Process

- **Job**: A Spark job is triggered by an action (e.g., collect(), save()). A job is a complete unit of work.
- **Stage**: A stage is a set of transformations that can be pipelined together without shuffling. The Spark job is divided into stages based on shuffle operations.
- **Task**: A task is the smallest unit of work and corresponds to a partition of data. Tasks within a stage are executed in parallel.

13. df to Spark SQL – Convert DataFrame Queries to SQL

To convert a DataFrame to Spark SQL, first register the DataFrame as a temporary view:

```
df.createOrReplaceTempView("table_name")
```

Then, you can query the DataFrame using Spark SQL:

```
result = spark.sql("SELECT * FROM table_name WHERE column_name > 10")
```

14. Job Cluster vs Interactive Cluster – Differences and When to Use

- **Job Cluster**:

  Definition: A cluster created to run specific jobs or batch jobs. It is terminated once the job completes.

  Use Case: Use when running batch jobs or scheduled jobs that don't require an interactive session.

- **Interactive Cluster**:

  Definition: A cluster that remains active for a longer period, allowing users to run interactive queries and notebooks.

  Use Case: Use when performing interactive analysis or debugging in notebooks.

15. Delta Table Features – Explain Z-ordering and Time Travel

- **Z-ordering**:

  Definition: Z-ordering is a technique to optimize the performance of range queries in Delta Lake by colocating related data in the same file.

  Example: You can Z-order a table by a column (e.g., customer_id).

```
deltaTable.optimize().zOrderBy("customer_id")
```

- **Time Travel**:

  Definition: Time travel in Delta Lake allows you to query a previous version of a table.

  Example:

```
df = spark.read.format("delta").option("timestampAsOf", "2023-01-01").load("/path/to/de
```

Glassdoor Datametica Review –

https://www.glassdoor.co.in/Reviews/Datametica-Reviews-E1011460.htm

Datametica Careers –

https://www.datametica.com/careers/

Subscribe to my YouTube Channel for Free Data Engineering Content –

https://www.youtube.com/@shubhamwadekar27

Connect with me here –

https://bento.me/shubhamwadekar

Checkout more Interview Preparation Material on –

https://topmate.io/shubham_wadekar