COMPANYWISE Q&A DELOITTE

DEVIKRISHNA R

LinkedIn: @Devikrishna R Email: visionboard044@gmail.com

1.Question: How do you find the missing numbers in a sequence of IDs in PySpark?

Answer:

To find missing numbers:

1. **Determine Range:** Identify the minimum and maximum values in the column.
2. **Generate Full Sequence:** Create a DataFrame with the full range of numbers.
3. **Find Missing Values:** Use an anti-join to find numbers in the full range that are not in the original column.

Code Implementation:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.sql.types import IntegerType

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

# Input Data
data = [(1,), (2,), (3,), (6,), (7,), (8,)]
df = spark.createDataFrame(data, ["id"])
```

```
# Determine range of IDs
min_id, max_id = df.agg({"id": "min"}).collect()[0][0], df.agg({"id":
    "max"}).collect()[0][0]

# Create full range DataFrame
full_range = spark.createDataFrame([(x,) for x in range(min_id,
    max_id + 1)], ["id"])

# Find missing IDs
missing_ids = full_range.subtract(df)

# Show results
missing_ids.orderBy("id").show()
```

Output:

```
+---+
| id|
+---+
|  4|
|  5|
+---+
```

---

2. Question: How do you select every nth (e.g., 3rd) row from a dataset in PySpark?
Answer:
To select every nth row:

1. **Add a Row Index:** Use row_number() to assign an index to each row.
2. **Filter Rows:** Use the modulo operation to filter rows whose indices are divisible by n.

Code Implementation:

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

# Input Data
data = [
    (1001, "John Doe", 50000),
    (2001, "Jane Smith", 60000),
    (1003, "Michael Johnson", 75000),
    (4000, "Emily Davis", 55000),
    (1005, "Robert Brown", 70000),
    (6000, "Emma Wilson", 80000),
    (1700, "James Taylor", 65000),
    (8000, "Olivia Martinez", 72000),
    (2900, "William Anderson", 68000),
    (3310, "Sophia Garcia", 67000)
]
df = spark.createDataFrame(data, ["emp_id", "name", "salary"])
```

```python
# Add row number
window_spec = Window.orderBy("emp_id")
indexed_df = df.withColumn("row_number", row_number().over(window_spec))

# Filter every 3rd row
nth_rows = indexed_df.filter((col("row_number") % 3) == 0).drop("row_number")

# Show results
nth_rows.show()
```

Output:

```
+-------+----------------+------+
| emp_id|            name|salary|
+-------+----------------+------+
|   1003| Michael Johnson| 75000|
|   6000|     Emma Wilson| 80000|
|   2900|William Anderson| 68000|
+-------+----------------+------+
```

---

1. Question: How do you design an incremental pipeline in ADF?
Answer:

- **Identify Changes:** Use a watermark column like LastModifiedDate or IncrementalID in your source data.
- **Configure Source Query:** Filter the source data using the watermark column (SELECT * FROM table WHERE LastModifiedDate > @Watermark).
- **Store Watermark:** Use an ADF variable or a dedicated table to store the latest processed watermark.
- **Update Watermark:** After successful ingestion, update the watermark for the next run.

---

2. Question: How do you configure ADF to send an email notification after the pipeline run?
Answer:

1. **Use Azure Logic Apps:**
    - Create a Logic App that sends an email using services like Outlook or SMTP.

- Trigger the Logic App via ADF Web Activity after pipeline completion.

2. **Add Success/Failure Paths:**
   - Use Success and Failure dependencies to trigger respective Web Activities or Logic Apps.

Steps:

- Configure Web Activity to call the Logic App.
- Pass the pipeline status (Success or Failure) as a parameter to the Logic App.

---

3. Question: How do you optimize a Copy Activity in ADF that runs for more than 6 hours?
Answer:

1. **Enable Parallelism:**
   - Set Degree of Parallelism in the Copy Activity.
2. **Use Staging:**
   - Enable staging (e.g., Azure Blob Storage or Azure Data Lake) for large data transfers.
3. **Partition Data:**
   - Use source and sink partitions to parallelize the data transfer.
4. **Optimize Source Query:**
   - Ensure indexes are used and queries are optimized.
5. **Adjust Integration Runtime:**
   - Scale up the Integration Runtime (IR) or use an Azure-Managed IR with more resources.

4. Question: How do you debug failed pipelines and resume execution from the failed activity?
Answer:

1. **Enable Activity Retry:**
   - Configure the Retry Policy in activities to handle transient failures.
2. **Debug Logs:**
   - Review activity logs, input/output datasets, and service logs for detailed error information.
3. **Resume Execution:**
   - Enable **Checkpoints** by configuring the Retry From Failure option in pipeline runs. This ensures the pipeline starts from the failed activity.

---

5. Question: How do you integrate ADF with GitHub/Azure DevOps and promote pipelines across environments?
Answer:

1. **Integration with Git:**
   - Connect ADF with a Git repository (Azure DevOps or GitHub) via the Manage tab.
   - Use feature branches for development.
2. **Export and Deploy:**
   - Export ARM templates from the source environment using ADF.
   - Use Azure Pipelines to deploy the ARM templates to target environments.

3. **CI/CD Pipeline:**
    - Use Azure DevOps to create build and release pipelines that include the ARM template deployment.

---

6. Question: What are the types of Integration Runtime, and when should you use Self-hosted IR?
Answer:

- **Azure IR:** For cloud-based data movement and transformation.
- **Self-hosted IR:** For on-premises or virtual network data sources. Use when:
    - Data resides in private networks.
    - You need access to on-premises databases/files.
- **SSIS IR:** For running SQL Server Integration Services (SSIS) packages in the cloud.

---

7. Question: What are the different types of triggers in ADF?
Answer:

1. **Schedule Trigger:** Executes pipelines at defined intervals or schedules.
2. **Tumbling Window Trigger:** Processes data in recurring time windows with a dependency on data availability.
3. **Event-based Trigger:** Executes pipelines in response to events like file arrival in Blob Storage or Data Lake.

Spark-Related Questions and Answers

1.Question: What optimization techniques have you used in your project?

Answer: Here are the key Spark optimization techniques:

1. **Caching and Persistence:**
   - Use .cache() or .persist() for repeated access to intermediate results.
   - Avoid caching large datasets unless required.
2. **Partitioning:**
   - Optimize the number of partitions using .repartition() or .coalesce().
   - Use partitioning to reduce data shuffling for large datasets.
3. **Broadcast Variables:**
   - Use broadcast() to share small lookup tables across executors.
4. **Optimize Joins:**
   - Use broadcast joins for small datasets.
   - Apply bucketing or sorting to reduce shuffle during large joins.
5. **Avoid Wide Transformations:**
   - Prefer narrow transformations (e.g., map, filter) over wide transformations (e.g., groupBy, reduceByKey).
6. **Predicate Pushdown:**
   - Ensure that filters are pushed to the source system to minimize data transfer.
7. **Skew Handling:**

    - Detect and handle data skew using salting or custom partitioning.

---

2. Question: What are common causes of OOM errors in Spark, and how do you address them?
Answer:

- **Causes:**
    1. Too many tasks running simultaneously, exceeding executor memory.
    2. Large joins or aggregations causing memory spikes.
    3. Inefficient use of caching, resulting in memory overflow.
    4. Data skew leading to uneven task distribution.
- **Solutions:**
    1. **Optimize Resource Allocation:**
        - Increase executor memory and cores using spark.executor.memory and spark.executor.cores.
    2. **Optimize Partitions:**
        - Use repartition() to balance data distribution and avoid memory overload.
    3. **Broadcast Joins:**
        - Use broadcast() for smaller datasets in joins.
    4. **Garbage Collection:**
        - Configure JVM settings (spark.memory.fraction and spark.memory.storageFraction).
    5. **Use Disk Spill:**

        - Allow Spark to spill data to disk by adjusting spark.sql.autoBroadcastJoinThreshold and spark.shuffle.spill.

3.Question : How does Spark calculate the initial number of partitions?

Answer:

- **RDDs:**
    - For HDFS files, the initial number of partitions is based on the file block size.
    - Number of partitions = total file size / HDFS block size.
- **DataFrames/Datasets:**
    - For DataFrames, Spark uses the configuration parameter spark.sql.shuffle.partitions (default: 200).
- **Custom Configurations:**
    - You can set the number of partitions explicitly using .repartition() or .coalesce().

---

4. Question: Why is count a transformation when used with groupBy and an action otherwise?

Answer:

- **With groupBy:**
    - groupBy creates a new RDD with grouped data. When count is applied, it results in another RDD transformation where counts for each group are computed.
- **Without groupBy:**

   - count directly triggers an action to compute the total number of rows in the dataset.

---

5. Question: What are Unity Catalog, Delta Live Tables, and AutoLoader in Databricks?
Answer:

- **Unity Catalog:**
   - A unified data governance solution for managing permissions, lineage, and auditing in Databricks.
   - Key Features: Fine-grained access control, centralized metadata, and data discovery.
- **Delta Live Tables (DLT):**
   - A declarative framework for building reliable data pipelines on Delta Lake.
   - Key Features: Simplifies ETL with SQL/Python, automatic quality checks, and error handling.
- **AutoLoader:**
   - A file ingestion utility for streaming and batch ingestion in Delta Lake.
   - Key Features: Schema inference, file notifications, and incremental loading.

---

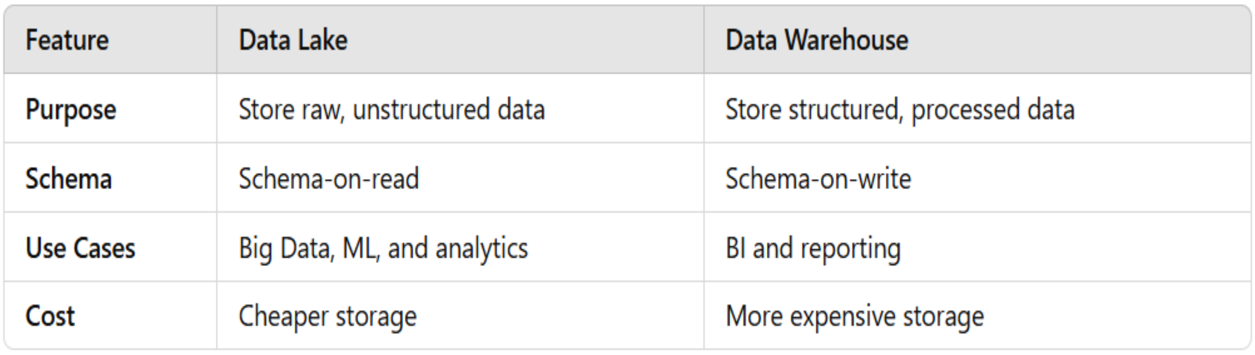
6. Question: How do you optimize workloads in Databricks?
Answer:

1. **Cluster Configuration:**
   - Use autoscaling for dynamic resource allocation.
   - Configure appropriate node types for workloads.

2. **Delta Lake Optimization:**
    - Optimize Delta Lake tables with OPTIMIZE and ZORDER BY.
    - Use VACUUM to clean up stale files.
3. **Caching:**
    - Use spark.conf.set("spark.sql.inMemoryColumnarStorage.compressed", true) for in-memory table caching.
4. **Query Optimization:**
    - Leverage Adaptive Query Execution (AQE).
    - Tune spark.sql.shuffle.partitions based on data size.
5. **Data Skew Handling:**
    - Detect skew with metrics, and use skew join optimization.

---

7. Question: What are key differences between a Data Lake and a Data Warehouse?
Answer:



Feature	Data Lake	Data Warehouse
Purpose	Store raw, unstructured data	Store structured, processed data
Schema	Schema-on-read	Schema-on-write
Use Cases	Big Data, ML, and analytics	BI and reporting
Cost	Cheaper storage	More expensive storage