INTERVIEW QUESTION & ANSWERS [HEXAWARE]

DEVIKRISHNA R

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

1. IR Types and When to Use Each Type

Integration Runtime (IR) is the compute infrastructure used by Azure Data Factory (ADF). The types of IR and their usage are:

- **Auto-Resolve IR:**
  - Default runtime for activities running in the same region as the Data Factory.
  - **Use Case:** When working with cloud data in the same region.
- **Self-Hosted IR:**
  - Used to access on-premises data or data in private networks.
  - **Use Case:** For hybrid data movement, connecting on-premises databases, or VNet.
- **Azure IR:**
  - For cloud-based data transformation and integration.
  - **Use Case:** For data flows, cloud-to-cloud data movement, or built-in data transformations.
- **SSIS IR:**
  - Specifically for running SSIS packages in ADF.
  - **Use Case:** When migrating existing SSIS workloads to Azure.

---

2. What is Copy Activity, and Why is it Used?

Copy Activity is used to move data from a source to a destination.

- **Purpose:**
  - Data ingestion and movement between different storage systems.
  - Supports basic transformations like mapping, filtering, and format conversions.
- **Use Case:**
  - Moving data from Blob Storage to SQL databases.
  - Loading large datasets efficiently from one source to another.

---

3. Lookup Activity: Why is it Used, and What Can You Do With It? Can You Update Using Lookup?

- **Why Use Lookup Activity?**
  - Retrieve data or values from a source dataset.
  - Used to fetch configurations or parameters dynamically.
- **What You Can Do:**
  - Fetch values for pipeline parameters.
  - Use fetched data in conditional flows or subsequent activities.
- **Can You Update Using Lookup?**
  - No, Lookup Activity is read-only. Updates require a Stored Procedure or Data Flow activity.

4. Difference Between Copy Activity and Data Flow

Feature	Copy Activity	Data Flow
Purpose	Data movement	Data transformation
Complex Transformations	Limited (e.g., column mapping)	Supports joins, aggregations, etc.
Performance	Lightweight	Uses Spark clusters for large-scale processing
Why Use Data Flow?	When complex transformations are required.	Use when queries in Copy Activity aren't sufficient.

---

5. In Blob Storage, You Have 1 Million Rows and Need to Delete Rows Starting with "A." How Do You Do It?

Solution:

- Use Azure Data Flow for transformation:
    1. Source: Connect to Blob Storage.
    2. Add a filter transformation:

       ```
       !startswith(column_name, 'A')
       ```

    3. Sink: Write the filtered data back to Blob Storage.

6.Write a Query to Separate Numbers and Letters from a Table

Column
1
2
3
A
C

SQL Query:

```sql
SELECT column AS numbers FROM table_name WHERE ISNUMERIC(column) = 1;
SELECT column AS letters FROM table_name WHERE ISNUMERIC(column) = 0;
```

---

7. Pass Column Name and Table Name Dynamically in a Query

Solution (Python):

```python
table_name = "my_table"
column_name = "my_column"

query = f"SELECT {column_name} FROM {table_name}"
df = spark.sql(query)
df.show()
```

---

8. Steps to Improve Pipeline Optimization if It Is Taking Too Long

1. **Partitioning and Distribution:** Use partitioned data for parallel processing.
2. **Parallelism:** Increase concurrency in activities.
3. **Filter Early:** Apply filters at the source to reduce data volume.
4. **Staging:** Use staging areas for intermediate results.
5. **Integration Runtime Optimization:** Use appropriate IR for workload locality.
6. **Query Optimization:** Optimize SQL queries in Copy Activity or Data Flow.

---

9. How to Implement Parallel Processing of Activities in ADF

Solution:

- **Steps:**
    1. Place multiple activities in parallel in the pipeline canvas.
    2. Ensure no dependency links between activities.
    3. Set the pipeline's concurrency level to support parallel execution.

Example: Parallel Copy Activities:

```
"activities": [
    {
        "name": "CopyActivity1",
        "type": "Copy",
        "inputs": [...],
        "outputs": [...]
    },
    {
        "name": "CopyActivity2",
        "type": "Copy",
        "inputs": [...],
        "outputs": [...]
    }
]
```

ROUND 2

1. How do you merge df1 and df2?

Answer:
Merge df1 and df2 based on the common column (e.g., Name or Gender):

```
from pyspark.sql import SparkSession

# Create DataFrames
df1 = spark.read.csv("path/to/df1.csv", header=True, inferSchema=True)
df2 = spark.read.csv("path/to/df2.csv", header=True, inferSchema=True)

# Merge DataFrames
merged_df = df1.join(df2, on=["Name", "Gender"], how="inner")
merged_df.show()
```

---

2. Find the 3rd highest salary in each department.

Answer:

```
WITH ranked_salaries AS (
    SELECT dept, sal,
           DENSE_RANK() OVER (PARTITION BY dept ORDER BY sal DESC) AS rank
    FROM emp
)
SELECT dept, sal
FROM ranked_salaries
WHERE rank = 3;
```

---

3. How do you get odd-numbered records by creating an ID in a table without an existing ID?

Answer:
Add a row ID using the monotonically_increasing_id() function and filter for odd IDs:

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

df_with_id = df.withColumn("row_id", monotonically_increasing_id())
odd_rows = df_with_id.filter(col("row_id") % 2 != 0)
odd_rows.show()
```

---

4. How did you implement Databricks in your project?

Answer:

- **Ingestion:** Used Databricks Auto Loader for incremental file ingestion from Azure Blob Storage.
- **Transformation:** Performed data cleaning, joins, and aggregations using PySpark.
- **Storage:** Saved data in Delta format for ACID transactions.
- **Orchestration:** Scheduled notebooks using Databricks Jobs or Azure Data Factory.
- **Analysis:** Integrated Databricks SQL for reporting and analytics.

---

5. What is SCD, and how did you implement SCD Type 2 in Databricks?

Answer:

- **Slowly Changing Dimensions (SCD):** A technique to track changes in dimension data.
- **SCD Types:**
  - **Type 1:** Overwrite existing records.
  - **Type 2:** Maintain historical data with effective and expiry dates.
  - **Type 3:** Add new columns for changes.

SCD2 Implementation in Databricks:

```
from pyspark.sql.functions import lit, current_date

# Update active flag for old records
updated_records = df_full.join(df_incremental, "id", "inner") \
                         .filter("df_full.col1 != df_incremental.col1") \
                         .withColumn("Active_Flag", lit("N")) \
                         .withColumn("To_Date", current_date())

# Add new records
new_records = df_incremental.withColumn("Active_Flag", lit("Y")) \
                            .withColumn("From_Date", current_date())

final_df = updated_records.union(new_records)
final_df.write.format("delta").save("path_to_delta_table")
```

---

6. What are the cluster types in Databricks, and where do you use each type?

Answer:

1. **All-Purpose Clusters:** For ad-hoc analysis and collaboration.
2. **Job Clusters:** For running scheduled or automated jobs.
3. **High-Concurrency Clusters:** For shared usage with fine-grained access control.
4. **Interactive Clusters:** For exploratory data analysis and development.

---

7. How did you get data from your source and implement it in notebooks?

Answer:

- **Data Ingestion:** Used Auto Loader, JDBC connections, or REST APIs to fetch data from sources like Azure Blob Storage or SQL databases.
- **Notebook Implementation:** Data was transformed using PySpark, with steps like cleaning, filtering, and aggregation, followed by saving in Delta format.

---

8. Difference Between Delta and Parquet Files

Feature	Delta	Parquet
ACID Support	Yes	No
Schema Evolution	Supported	Limited
Time Travel	Yes	No
Performance	Optimized with indexing (Z-Ordering)	Good, but no advanced indexing

---

9. Why use Databricks when you have ADF?

Answer:

- **Databricks:** Best for big data transformations, machine learning, and advanced analytics.
- **ADF:** Primarily for orchestration and simple data movements.
- **Why Databricks?** For complex transformations, distributed processing, and real-time streaming.

---

10. Why use Window Functions, and what are they?

Answer:

- **Why:** To perform calculations across a set of rows related to the current row.
- **Common Functions:**
    - ROW_NUMBER(): Assigns a unique number to each row.
    - DENSE_RANK(): Assigns ranks without gaps.
    - SUM() OVER(): Computes cumulative sums.

Example:

```
SELECT dept, sal, ROW_NUMBER() OVER (PARTITION BY dept ORDER BY sal DESC) AS row_num
FROM emp;
```

---

11. Merge Two DataFrames (df1 and df2)

Answer:

```
df1 = spark.read.csv("path1.csv", header=True)
df2 = spark.read.csv("path2.csv", header=True)

merged_df = df1.union(df2)
merged_df.show()
```

---

12. Does UNION Remove Duplicates in PySpark? If Not, How Do You Remove Them?

Answer:

- **No**, union() does not remove duplicates.
- Use distinct() to remove duplicates:

```
union_df = df1.union(df2).distinct()
union_df.show()
```
