**Deloitte USI Data Engineer Interview Guide – Experienced 3+**

**Technical round 1 and 2 combined**

**1. Full Load vs Incremental Load in ADF**

 **Full Load**:

**Definition**: Full load refers to the process of copying all data from the source to the target system, replacing any existing data in the target.

**Example**: In PySpark, you might overwrite the existing data in the target with fresh data from the source:

data\_df.write.format("csv").mode("overwrite").save("/path/to/your/target/table")

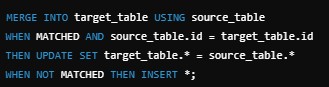
**Use Case**: Used when you need to reload all data from the source, especially in cases where there are no mechanisms in place to track changes.

 **Incremental Load**:

**Definition**: Incremental load refers to copying only the new or updated data from the source to the target, avoiding the need to reload the entire dataset.

**Example**: In SQL, an incremental load can be achieved by using a MERGE

statement to synchronize the target with the changes in the source:



**Use Case**: Ideal for high-volume data transfers, as it only processes changed or new records, improving performance and reducing load time.

**2. ADF Optimization Techniques**

 **Optimize Data Movement**:

 Use staging in Copy Activity to temporarily store data before moving it to the final destination, especially for large datasets.

 Partitioning: Partition your data and use parallel copies for faster ingestion.

 **Data Flow Optimization**:

 Use filter pushdown to limit the data processed in the pipeline.

 Avoid unnecessary transformations in Data Flow. Use Projection Pushdown to reduce the data volume and improve performance.

 **Monitoring**:

 Implement monitoring and logging for ADF pipelines to identify bottlenecks and optimize performance.

**3. Notebook Optimization Strategies**

 **Data Caching**: Use .cache() or .persist() on frequently accessed datasets to avoid recomputing them.

df.cache()

 **Broadcast Joins**: For smaller tables, use broadcast joins to minimize shuffling. from pyspark.sql.functions import broadcast

df1.join(broadcast(df2), "key")

 **Partitioning**: Use optimal partitioning strategies to avoid skew in your data and ensure a balanced workload across the cluster.

 **Avoid Shuffling**: Minimize operations that require shuffling data, such as groupBy or join without partitioning, which can significantly degrade performance.

**4. Snowflake vs Star Schema**

 **Star Schema**:

**Structure**: A central fact table connected to multiple dimension tables.

**Example**: A Sales fact table connected to Product, Customer, and Time dimension tables.

**Use Case**: Simpler queries and better performance when querying dimensional data. Easier for beginners to design.

 **Snowflake Schema**:

**Structure**: Similar to Star Schema but with normalized dimension tables, breaking them into sub-dimensions.

**Example**: The Customer dimension table is broken into Customer, Region, and Country tables.

**Use Case**: More complex queries and better storage optimization but slower performance due to the need for multiple joins.

**5. Fact vs Dimension Tables**

 **Fact Tables**:

 **Definition**: Contain measurable, quantitative data about a business event or transaction (e.g., sales amount, transaction count).

 **Example**: Sales(order\_id, customer\_id, amount).

 **Dimension Tables**:

 **Definition**: Contain descriptive attributes related to the business dimensions (e.g., customer name, product type).

 **Example**: Customers(customer\_id, customer\_name, region).

**6. Adding a New Column in PySpark**



 This will add a new column called new\_column with the default value "default\_value" for all rows.

**7. Parallel Copies in ADF**

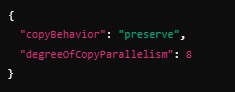
 **Prerequisites**:

The source data should be partitioned appropriately to support parallel processing.

Ensure the source data is large enough to benefit from parallel copying (e.g., large databases or storage systems).

 **Configuration**:

Set the degreeOfCopyParallelism parameter in the Copy activity to define how many copies should run concurrently:



Use multiple parallel copy tasks when copying from multiple sources or to multiple targets for better performance.

**8. Synapse Analytics Features and Use Cases**

 **Synapse Analytics** combines big data and data warehousing for real-time analytics and reporting. It integrates seamlessly with Azure data lake, SQL Data Warehouse, and Power BI.

 **Key Features**:

SQL Pools: Dedicated resources for large-scale query processing.

Apache Spark Pools: For big data analytics and machine learning workflows. Real-time Analytics: Integrated support for streaming and batch data

processing.

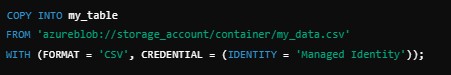
 **Use Cases**:

Data warehousing, business intelligence, big data analytics, real-time reporting, machine learning.

For personal use only. Redistribution or resale is prohibited. © Shubham Wadekar

**9. Data Load in Synapse Table**

To load data into a Synapse table, use the COPY command:



 **Optimization**: Use **batch loading** and partitioning strategies for optimal performance when loading large datasets.

**10. Activities in ADF**

 **Copy Activity**: For copying data from a source to a destination (e.g., SQL database to data lake).

 **Lookup Activity**: Retrieves values from a data source to use in subsequent activities.

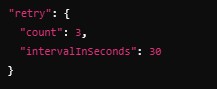
 **Data Flow Activity**: Used for data transformation with complex logic.

 **Execute Pipeline Activity**: To execute another pipeline from within a pipeline.

 **ForEach Activity**: Runs a series of activities for each item in a collection.

**11. Error Handling in ADF**

 Retry Policies: You can define retry policies for activities to handle transient errors:



 Try-Catch Mechanism: Use the Until activity for error handling and retries.

 Logging: Use the Web Activity or Azure Monitor to log and monitor errors.

**12. Data Flow Service in ADF**

Data Flows in ADF are used to define and execute data transformations visually. It allows you to build complex data transformation logic without writing code.

 **Components**: Source, Transformation (e.g., aggregate, join), Sink (e.g., SQL Server, Azure Blob Storage).

 **Use Case**: When you need to transform data before loading it into a data warehouse or lake.

**13. How to Check Spark Version**

To check the version of Spark being used in your environment, simply run the following command:

print(spark.version)

This will return the version of Apache Spark that is currently in use.

**14. CI/CD Tools – Jenkins and Azure DevOps**

 **Jenkins**:

 Open-source tool for automating the build, test, and deployment of applications.

 Supports integration with various version control systems and can trigger automated builds on code changes.

 **Azure DevOps**:

 Cloud-based suite for continuous integration and continuous deployment

(CI/CD).

 Provides tools for version control (Git), build automation, release management, and project management.

**15. Where to Store Secret Keys**

Use Azure Key Vault for securely managing secrets such as API keys, passwords, and connection strings:

 Azure Key Vault allows secure, controlled access to secrets using managed identities and RBAC.

 Example for accessing secrets from Key Vault using Python:

from azure.identity import DefaultAzureCredential from azure.keyvault.secrets import SecretClient

# Create a client using managed

identity credential = DefaultAzureCredential()

client = SecretClient(vault\_url="https://<your-keyvault-name>.vault.azure.net/", credential=credential)

# Get a secret

secret = client.get\_secret("my-secret")

print(secret.value)