AZURE DATA ENGINEERING INTERVIEW Q & A - TOPICWISE

(This document contains interview questions which I experienced personally in all my 20+ interviews)

Skill Covered

- Pyspark
- SQL
- Databricks
- Azure Data Factory
- ADLS
- Python
- Data Modeling and Architecture
- Other Questions(Azure Devops, Azure functions, LogicApps and general ques)

1. **Pyspark**

- Explain lazy evaluation in PySpark.
- Difference between groupByKey and reduceByKey.
- What is the difference between repartition and coalesce?
- Explain broadcast join in PySpark.
- How to register a User Defined Function (UDF) in PySpark?
- What is the purpose of SparkContext and SparkSession?
- How does caching work in PySpark?
- Difference between wide and narrow transformations.
- Write a PySpark code to:
    1. Read a CSV file,
    2. Join two DataFrames,
    3. Perform aggregation,
    4. Filter rows based on a condition.
- How to create a rank column using the Window function in PySpark?
- What is Z-ordering in Spark?
- Explain Delta Lake and its benefits over Parquet.
- What is AQE (Adaptive Query Execution) in Databricks?
- How to handle incremental load in PySpark when the table lacks a last_modified or updated_time column?
- How many jobs, stages, and tasks are created during a Spark job execution?
- How to persist and cache data in PySpark?
- How to handle null values in PySpark (drop/fill).
- Difference between Spark SQL and PySpark DataFrame APIs.
- DataFrame API syntax to filter and aggregate data: Example: df.groupBy("department").agg(_sum("salary").alias("sumofsalary")).
- How do you design and implement data pipelines using Azure Data Factory?
- Explain the use of Azure Synapse Analytics and how it integrates with other Azure services.
- How do you optimize data storage and retrieval in Azure Data Lake Storage?
- How do you handle schema evolution in Azure Data Lake?
- Describe the process of setting up and managing an Azure SQL Database.

- Explain the concept of PolyBase in Azure SQL Data Warehouse.
- How do you implement security measures for data in transit and at rest in Azure?
- How do you use Azure Stream Analytics for real-time data processing?
- What are the different data partitioning strategies in Azure SQL Data Warehouse?
- How do you monitor and troubleshoot data pipelines in Azure Data Factory?
- Describe the process of integrating Azure Databricks with Azure Data Lake Storage.
- How do you manage and automate data workflows using Azure Logic Apps?
- Explain the concept of Managed Identity in Azure and its use in data engineering.
- How do you ensure data consistency and reliability in distributed systems on Azure?
- Describe the process of performing ETL operations using Azure Data Factory.
- How do you use Azure Monitor to track and analyze performance metrics of your data infrastructure?
- Explain the role of Azure Key Vault in securing sensitive data.
- How do you handle big data processing using Azure HDInsight?
- What are the best practices for data archiving and retention in Azure?
- How do you implement disaster recovery and backup strategies for data in Azure?

2. SQL

- How to find the second-highest salary in a table? (Provide multiple solutions).
- Difference between TRUNCATE and DELETE.
- Difference between stored procedures and functions.
- What are views and CTEs (Common Table Expressions)?
- How do aggregation functions differ from analytic functions in SQL?
- Write a SQL query to:
    1. Find top 3 earners in each department.
    2. Get the count of duplicates in a table.
    3. Use LAG and RANK functions in a query.
    4. Count occurrences of each word in a table column.
- What is the difference between star schema, snowflake schema, and 3NF?
- Explain ACID properties in SQL with an example.
- Write a SQL query to convert row-level data to column-level data using pivot.
- How to create and use indexes (clustered vs non-clustered).
- Common queries for optimization:
    1. Find records that exist in one table but not in another.
    2. Write SQL to find employees earning more than their manager.
- Union vs Union All: Differences in usage and performance.
- Concurrency issues in SQL: How do you resolve them?
- Explain the role of Azure SQL Database in cloud-based data solutions.
- How do you set up and configure an Azure Data Factory pipeline for ETL processes?
- What are the best practices for database security in Azure SQL Database?
- How do you implement data partitioning in Azure SQL Data Warehouse?
- Describe the use of Azure Cosmos DB and its benefits over traditional SQL databases.
- How do you integrate Azure SQL Database with other Azure services?
- Explain the concept of Azure Data Lake and its integration with SQL-based systems.
- How do you use Azure Logic Apps to automate data workflows in SQL databases?
- Describe the process of migrating on-premises databases to Azure SQL Database.
- How do you use Azure Data Lake Analytics with U-SQL for big data processing?

- Explain the differences between Azure SQL Database and Azure SQL Managed Instance.
- How do you ensure high availability and disaster recovery for Azure SQL databases?
- What are the common challenges in managing large-scale SQL databases in Azure?
- How do you optimize query performance in Azure SQL Database?
- Describe the process of setting up and managing an Azure Synapse Analytics workspace.
- How do you monitor and troubleshoot issues in Azure SQL Database?
- Explain the use of Azure Active Directory for SQL database authentication.
- How do you implement data encryption in Azure SQL Database?
- Describe the process of using Azure SQL Data Sync for data replication.
- What are the key considerations for designing a scalable data architecture in Azure?

SQL Important coding ques:

- Find the second-highest salary in a table.
- Write a SQL query to find the top 3 earners in each department.
- Get the count of duplicates in a table.
- Use LAG and RANK functions in a query.
- Count occurrences of each word in a table column.
- Write a SQL query to convert row-level data to column-level data using pivot.
- Find records that exist in one table but not in another.
- Write SQL to find employees earning more than their manager.
- Write a SQL query to display the cumulative sum of a column.
- Write a SQL query to find the nth highest salary in a table.
- Write a SQL query to list all employees who joined in the last 6 months.
- Write a SQL query to get the department-wise highest salary.
- Write a SQL query to join two tables and display specific columns.
- Write a SQL query to find gaps in a sequence of numbers.
- Write a SQL query to remove duplicate rows from a table.
- Write a SQL query to find the average salary of employees by department.
- Write a SQL query to calculate the median salary of employees.
- Write a SQL query to rank employees based on their performance score.
- Write a SQL query to find employees who do not have a manager.
- Write a SQL query to create a new table with the results of a complex query.

3. Databricks

- What is the difference between a job cluster and an interactive cluster?
- Explain Databricks Lakehouse architecture (Bronze, Silver, Gold layers).
- How to copy files from Blob Storage to Databricks?
- How to create and deploy notebooks in Databricks?
- What is Unity Catalog in Databricks?
- How to run one notebook from another notebook using %run or dbutils.notebook.run()?
- How to connect ADLS (Azure Data Lake Storage) to Databricks?
- What are Delta logs, and how to track data versioning in Delta tables?
- How to implement SCD Type 1 and Type 2 using PySpark in Databricks?
- How do autoscaling clusters work in Databricks?
- Explain Databricks optimization techniques (e.g., file format, partitioning, caching).
- How to monitor Databricks jobs?
- Explain the difference between caching and persisting in Databricks.
- How to give permission to specific notebooks or clusters to other users.
- How to implement data validation and quality checks in Databricks?
- How to manage and automate ETL workflows using Databricks Workflows?
- What are the best practices for securing data in Databricks?
- How to integrate Databricks with Azure DevOps for CI/CD pipelines?
- How to use Databricks Delta Live Tables for continuous data processing?

4. **Azure Data Factory (ADF)**

- How to implement incremental load in ADF?
- How do full loads differ from incremental loads in ADF?
- What are the types of Integration Runtimes (IR) in ADF?
- Explain the process of copying large files (e.g., 3TB) from on-premises to Azure in ADF.
- How to handle error handling in ADF using retry, try-catch blocks, and failover mechanisms?
- What is the binary copy method, and when is it used?
- How to perform parallel copies in ADF using partitioning?
- How do you handle complex ADF pipelines? Explain challenges faced.
- What are the activities in ADF (e.g., Copy Activity, Notebook Activity)?
- How to pass parameters from ADF to Databricks notebooks?
- How to connect to Salesforce using ADF securely?
- Where to store sensitive information like passwords in ADF (e.g., Azure Key Vault)?
- How to copy all tables from one source to the target using metadata-driven pipelines.
- Trigger types in ADF (Schedule, Tumbling Window, etc.).
- How to implement error retry policies in ADF pipelines?
- How to implement data validation and data quality checks in ADF?
- How do you optimize the performance of ADF pipelines?
- Describe the process of integrating ADF with Azure Synapse Analytics.
- How do you handle schema drift in ADF?
- Explain how to use ADF with Azure Databricks for complex transformations.
- How do you implement data masking in ADF for sensitive data?
- Describe the steps to migrate SSIS packages to ADF.
- How do you monitor and troubleshoot ADF pipeline failures?
- Explain the role of ADF in a hybrid data integration scenario.

5. Data Lake Storage (ADLS)

- Difference between Blob Storage and Azure Data Lake Storage (ADLS).
- Why is mounting preferred over using access keys to connect Databricks with ADLS?
- How do you ensure security when connecting Databricks with ADLS?
- Explain how to copy all files with different formats (CSV, Parquet, Excel) from a source to target.
- How to track file names in the output table while performing copy operations in ADF?
- How to optimize data retrieval from ADLS in Spark?
- What are the security features available in ADLS (e.g., access control lists, role-based access)?
- How do you implement versioning in ADLS?
- Explain the use of hierarchical namespaces in ADLS.
- How do you manage data lifecycle policies in ADLS?
- What are the differences between ADLS Gen1 and Gen2?
- How do you monitor and troubleshoot issues in ADLS?
- How do you handle large-scale data ingestion into ADLS?
- Explain the process of setting up disaster recovery for ADLS.
- How do you integrate ADLS with Azure Synapse Analytics?
- How do you ensure data quality and validation in ADLS?
- Describe the process of setting up and managing access control lists (ACLs) in ADLS.
- How do you implement data encryption at rest and in transit in ADLS?
- How do you integrate ADLS with Azure Databricks for data processing?
- What are the best practices for managing and optimizing storage costs in ADLS?
- How do you handle schema evolution in ADLS?
- Explain the process of auditing and logging access to data in ADLS.

6. Python

- Write Python code to:
    - Generate prime numbers.
    - Check if a string is a palindrome.
    - Replace vowels in a string with spaces.
    - Count word occurrences in a string.
    - Find consecutive numbers in a list (e.g., "Print the number that appears consecutively 3 times").
- How do you use dictionaries in Python to store key-value pairs efficiently?
- Write Python code to split a name column into firstname and lastname.
- Generate Fibonacci numbers.
- Identify duplicates in a list and count their occurrences.
- Explain the difference between lists, tuples, and sets in Python.
- How do you handle missing or null values in a dataset using Python?
- Describe the process of data serialization and deserialization in Python.
- How do you read and write data to and from a database using Python?
- Explain the concept of list comprehensions and provide an example.
- How do you optimize Python code for better performance?
- Describe the use of context managers in Python.
- How do you handle exceptions and errors in Python?
- Explain the role of pandas library in data manipulation.
- How do you integrate Python with big data tools like Apache Spark?

7. Data Modeling and Architecture

- Difference between a database, data warehouse, and data lake.
- What are fact and dimension tables in data modeling?
- Difference between ER modeling and dimensional modeling.
- What is the process of normalization, and why is it required?
- Explain the deployment architecture of a data pipeline involving ADF, Databricks, and ADLS.
- Build a simple data warehouse for a grocery store.
- What are the different types of data schemas, and how do you choose the right one for your data model?
- How do you handle slowly changing dimensions (SCD) in data modeling?
- Explain the concept of denormalization and when it should be used.
- How do you design a star schema for a sales reporting system?
- What is data mart, and how does it differ from a data warehouse?
- How do you ensure data consistency and integrity in a distributed data architecture?
- Describe the role of metadata in data modeling and data architecture.
- How do you optimize data models for performance and scalability?
- Explain the use of surrogate keys in dimensional modeling.
- How do you implement a data governance framework in a data lake environment?

8. Other Questions

- What is DAG in Spark, and how does it work?
- Explain Spark architecture (Driver vs Worker).
- What is the difference between wide transformations and narrow transformations in Spark?
- How does fault tolerance work in ADF and Databricks?
- What challenges have you faced in managing large datasets (e.g., 3TB+ files)?
- How do you implement CI/CD pipelines for deploying ADF and Databricks solutions?
- How to improve the performance of a Spark job?
- What is the use of Delta Lake, and how does it support ACID transactions?
- What is a Unity Catalog in Databricks?
- Explain the role of integration runtimes in hybrid scenarios (on-prem to cloud).
- How to design an end-to-end data pipeline architecture?
- Snowflake-specific features:
    - How does it compare to Delta Lake or other data lakes?
    - When would you choose Snowflake over other platforms?
- General Cloud Knowledge:

    - Difference between Azure Logic Apps, Azure Functions, and Azure Data Factory.
    - Azure Synapse Analytics: How does it compare to Databricks?

- How to use Azure DevOps for CI/CD pipelines.
- What are the key features of Azure DevOps?
- How do you implement continuous integration (CI) and continuous deployment (CD) in Azure DevOps?
- Explain the role of pipelines in Azure DevOps.
- How do you manage dependencies and versioning in Azure DevOps?
- What are Azure Functions, and how do they differ from traditional web services?
- How do you deploy and manage Azure Functions?
- Explain the use cases for Azure Functions in data engineering.
- How do you secure secrets and keys using Azure Key Vault?
- How do you integrate Azure Key Vault with other Azure services?
- What are the benefits of using Azure Synapse Analytics for big data processing?
- How does Azure Synapse integrate with other Azure services?
- Explain the difference between dedicated SQL pool and serverless SQL pool in Azure Synapse.
- How do you monitor and optimize performance in Azure Synapse?
- Describe the process of data ingestion in Azure Synapse.
- What are the best practices for data partitioning and distribution in Azure Synapse?
- How do you implement security and compliance in Azure Synapse?
- Explain the concept of data integration and transformation in Azure Data Factory.
- How do you handle error handling and retry mechanisms in Azure Data Factory?
- Describe the role of triggers and schedules in Azure Data Factory.
- How do you use Azure Monitor to track and analyze performance metrics in Azure services?

Pyspark

1. Explain lazy evaluation in PySpark.
Lazy evaluation means that **PySpark doesn't execute your transformations right away**. It just builds a logical plan (a **DAG**) until an **action** like .collect(), .show(), or .write() is called.

- **Why?**

This lets Spark optimize the execution — combining or reordering steps for better performance.

- **Example:**

```
df = spark.read.csv("data.csv")
filtered = df.filter(df.age > 30)   # Lazy
filtered.show()                     # Triggers actual execution
```

2. Difference between groupByKey() and reduceByKey()
Both are used for grouping data by key, but behave differently:

Feature	groupByKey()	reduceByKey()
Operation	Groups all values per key	Combines values per key using a reduce function
Performance	More **expensive** (shuffles all values)	**Efficient**, does partial reduction before shuffle
Use Case	When you just want grouped values	When you want to **aggregate** values (e.g., sum)

- Example:

```
# groupByKey
rdd.groupByKey()

# reduceByKey
rdd.reduceByKey(lambda a, b: a + b)
```

3. What is the difference between repartition() and coalesce()?

Feature	repartition()	coalesce()
Data Shuffle	Full shuffle	Minimal shuffle
Use Case	Increase or change partitions	Decrease number of partitions
Performance	**Slower**, expensive	**Faster**, optimized for shrinking

- Example:

```
df.repartition(10)   # Good before heavy operations
df.coalesce(2)       # Good before writing small number of files
```

4. Explain broadcast join in PySpark.
When one of your tables is **small**, Spark can **broadcast** it to all nodes to avoid shuffling the larger table — making the join much faster.

- Example:

```
from pyspark.sql.functions import broadcast

df1 = spark.read.csv("large_table.csv")
df2 = spark.read.csv("small_lookup.csv")

joined = df1.join(broadcast(df2), "key")
```

✓ Best used when the smaller DataFrame is under **10MB–100MB** depending on the cluster size.

5. How to register a User Defined Function (UDF) in PySpark?
A UDF lets you use **custom Python functions** in PySpark DataFrames.

- Example:

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

def to_upper(s):
   return s.upper()

to_upper_udf = udf(to_upper, StringType())

df = df.withColumn("upper_name", to_upper_udf(df.name))
```

6. What is the purpose of SparkContext and SparkSession?

- **SparkContext** is the **entry point** to Spark core – RDD-based APIs.
- **SparkSession** is the unified entry point to **DataFrame and SQL APIs**, introduced in Spark 2.x.

In PySpark, you typically use SparkSession (not SparkContext directly).

- Example:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("example").getOrCreate()
sc = spark.sparkContext  # Access SparkContext if needed
```

7. How does caching work in PySpark?

When you **cache** a DataFrame or RDD, Spark keeps it in **memory**, so if it's reused later, it doesn't recompute the whole transformation chain.

- Example:

```
df.cache()
df.count()  # This triggers the caching
```

✔ Ideal for iterative or repeated access to the same data.

8. Difference between wide and narrow transformations

Transformation Type	Example Functions	Data Movement	Triggers Shuffle?
Narrow	map(), filter(), flatMap()	No	No
Wide	groupBy(), reduceByKey(), join()	Yes (shuffle)	Yes

Narrow = each partition can be processed independently.
Wide = needs data from multiple partitions (slower and costlier).

9. PySpark code to:
a. Read a CSV file,
b. Join two DataFrames,
c. Perform aggregation,
d. Filter rows based on a condition:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum

spark = SparkSession.builder.appName("PySparkExample").getOrCreate()

# 1. Read CSV files
orders = spark.read.option("header", True).csv("orders.csv")
customers = spark.read.option("header", True).csv("customers.csv")

# 2. Join DataFrames on customer_id
joined_df = orders.join(customers, orders.customer_id == customers.customer_id, "inner")
```

```
# 3. Aggregate: total order amount per customer
agg_df = joined_df.groupBy("customer_id").agg(sum("order_amount").alias("total_spent"))

# 4. Filter: customers who spent more than 1000
result = agg_df.filter(agg_df.total_spent > 1000)

result.show()
```

10. How to create a rank column using the Window function in PySpark?

```
from pyspark.sql.window import Window
from pyspark.sql.functions import rank

windowSpec = Window.partitionBy("department").orderBy("salary")

df.withColumn("rank", rank().over(windowSpec)).show()
```

✔ You can also use dense_rank() or row_number() depending on the use case.

11. What is Z-ordering in Spark?

Z-ordering is a technique used in **Delta Lake** to **optimize data skipping** when reading.

- It sorts the data based on one or more columns using a **Z-order curve** (space-filling curve), clustering related data together.

Use case: Improves read performance for selective queries on columns like date, customer_id.

- Example:

```
df.write.format("delta").option("dataChange", "false").mode("overwrite").save("/path")

# Optimize with Z-order
spark.sql("OPTIMIZE delta.`/path` ZORDER BY (customer_id)")
```

12. Explain Delta Lake and its benefits over Parquet.

Delta Lake is an **open-source storage layer** that brings **ACID transactions**, **schema enforcement**, and **time travel** to your Parquet-based data lakes.

Feature	Delta Lake	Parquet
Transactions	ACID compliant	✕
Schema Evolution	Supported	Limited
Time Travel	Yes	✕
Data Updates/Delete	Native support via MERGE	✕ (Needs overwrite)
Performance Features	Z-order, file compaction, caching	Basic storage format

13. What is AQE (Adaptive Query Execution) in Databricks?

AQE allows Spark to **dynamically optimize** query plans at runtime based on the actual data characteristics.

- Key Features:
  - Dynamically switch join strategies (e.g., broadcast vs shuffle)
  - Skew join handling
  - Dynamic partition pruning
- Example:

```
spark.conf.set("spark.sql.adaptive.enabled", True)
```

✔ Helps handle data skew and improve performance significantly, especially for complex queries.

14. How to handle incremental load in PySpark when the table lacks last_modified or

updated_time?
If no timestamp column is available:
Approach 1: Hash-Based Comparison

- Add a **hash column** (e.g., MD5 on concatenated columns) to detect changes.
- Compare with previously stored hash to identify new/changed records.

Approach 2: Full load with Delta Merge

- Load the entire dataset and use MERGE INTO in Delta to upsert data.

```
# Pseudo Delta Merge code
MERGE INTO target USING source
ON target.id = source.id
WHEN MATCHED THEN UPDATE SET …
WHEN NOT MATCHED THEN INSERT …
```

15. How many jobs, stages, and tasks are created during a Spark job execution?

- **Job**: One job per **action** (e.g., .collect(), .write())
- **Stage**: Divided by **shuffle boundaries**
- **Task**: One task per **partition** in a stage

◈ Example:

```
df.groupBy("col").sum().show()
```

- Triggers **1 job**
- If 3 shuffles: **3 stages**
- If each stage has 4 partitions: **4 tasks per stage**

You can see this breakdown in **Spark UI** under the **Jobs** tab.

16. How to persist and cache data in PySpark?
Use .cache() to store in memory or .persist() for more control.

```
df.cache()       # Stores in memory (default)
df.persist()     # Can specify MEMORY_AND_DISK, etc.
```

☞ Always run an action (like .count()) after caching to trigger the actual storage.

17. How to handle null values in PySpark (drop/fill)?

- **Drop nulls**:

```
df.dropna()              # Drop rows with any nulls
df.dropna(subset=["col1"]) # Drop rows if col1 is null
```

- **Fill nulls**:

```
df.fillna("Unknown")              # Fill all nulls with "Unknown"
df.fillna({"age": 0, "name": "N/A"})   # Fill specific columns
```

18. Difference between Spark SQL and PySpark DataFrame APIs

Feature	Spark SQL	PySpark DataFrame API
Syntax Style	SQL-like (string-based queries)	Pythonic, object-oriented
Use Case	Good for those familiar with SQL	More flexible for programmatic operations
Example	spark.sql("SELECT * FROM employees")	df.select("*")
Performance	Both compile to the same logical plan	Same performance
Use in notebooks	Spark SQL is great for quick queries	PySpark is ideal for complex logic

☞ Under the hood, **both are optimized using Catalyst** and execute similarly.

19. DataFrame API syntax to filter and aggregate data:

```
from pyspark.sql.functions import sum as _sum

df.groupBy("department").agg(_sum("salary").alias("sumofsalary")).filter("sumofsalary >
```

50000").show()

- Explanation:
  - groupBy("department"): Groups rows by department
  - agg(_sum("salary")): Aggregates salary column
  - alias(...): Renames aggregated column
  - filter(...): Filters aggregated result

20. How do you design and implement data pipelines using Azure Data Factory (ADF)?
ADF Pipeline Design Flow:

1. **Source Identification**
   - Connectors: On-prem (SQL, Oracle), Cloud (ADLS, Blob, S3, API, etc.)
2. **Create Linked Services**
   - Define connections to source/destination.
3. **Datasets**
   - Represent data structures (tables/files).
4. **Pipeline Activities**
   - Use **Copy Activity**, **Data Flow**, **Stored Procedure**, **Notebook**, etc.
5. **Orchestration**
   - Use **Trigger** (schedule/event-based/manual)
   - Use **control flow**: If, ForEach, Until, etc.
6. **Monitoring**
   - Use **ADF Monitoring tab**, alerts, and logs.

- Example Use Case:
  - Extract from SQL → Transform in Data Flow → Load to ADLS/SQL DB.

21. Explain the use of Azure Synapse Analytics and how it integrates with other Azure services.
Azure Synapse Analytics is an integrated analytics service combining **big data and data warehousing**.

- **Features:**
  - SQL-based data warehouse (Synapse SQL)
  - Spark runtime for big data processing
  - Serverless and dedicated pools
  - Deep integration with Power BI, ADF, Azure ML, ADLS
- **Integration Examples:**
  - **ADLS Gen2**: Synapse reads data directly from lake
  - **ADF Pipelines**: Orchestrate data flows into Synapse
  - **Power BI**: Direct Query mode for reports
  - **Azure Purview**: Metadata catalog and lineage
  - **Azure Key Vault**: Secure credential storage

22. How do you optimize data storage and retrieval in Azure Data Lake Storage (ADLS)?
✓ **Best Practices for Optimization:**

1. **Partitioning**
   - Use date-based folder structure (e.g., /year/month/day/)
   - Helps improve query performance and scanning
2. **File Size Tuning**
   - Avoid small files (optimum: 100–250 MB per file)
3. **Format Choices**
   - Use columnar formats like **Parquet/ORC** over CSV
4. **Compression**
   - Use snappy/gzip for faster transfer and less storage
5. **Delta Lake Usage**
   - Enables ACID, upserts, schema evolution

6. **Z-Ordering**
    - For faster lookup of specific values
7. **Caching (Databricks)**
    - Cache frequently accessed data

23. How do you handle schema evolution in Azure Data Lake?
Delta Lake provides the best support for schema evolution in ADLS.
✅ **Handling Schema Changes:**

1. **Auto Merge Schema (Databricks)**:

```
spark.conf.set("spark.databricks.delta.schema.autoMerge.enabled", "true")
new_df.write.format("delta").mode("append").option("mergeSchema", "true").save("path")
```

2. **Versioning**:
    - Delta supports **time travel** to revert to older schema versions.
3. **Structured streaming with evolving schema**:
    - Use schema hints or define evolving schemas with caution.
4. **Monitoring schema drift**:
    - In **ADF Mapping Data Flows**, enable "Allow schema drift"

24. Describe the process of setting up and managing an Azure SQL Database
✅ **Setup Steps:**

1. **Create Azure SQL DB:**
    - From Azure Portal → Create Resource → Azure SQL → SQL Database
2. **Configure Settings:**
    - Choose server (or create new), pricing tier, backup, security
3. **Connect:**
    - Use **SSMS**, **Azure Data Studio**, or **ADF/Databricks**
4. **Firewall Settings:**
    - Allow your IP or use Azure services
5. **Create Tables/Objects:**
    - Run SQL scripts for schema setup
6. **Performance Tuning:**
    - Use **Query Performance Insight**, **Automatic Tuning**
7. **Security:**
    - Use **Transparent Data Encryption**, **Azure Defender**, **Key Vault**
8. **Monitoring:**
    - Use **Azure Monitor**, **Log Analytics**, **Alerts**

25. Explain the concept of PolyBase in Azure SQL Data Warehouse (now Synapse SQL)

PolyBase is a feature in **Azure Synapse Analytics** that lets you run T-SQL queries over **external data sources** like:

- Azure Data Lake Storage (ADLS)
- Blob Storage
- Hadoop
- External SQL Servers

📯 **Why is it useful?**

- Enables **ELT (Extract-Load-Transform)** pattern: load data directly from files into Synapse tables without pre-processing.
- You can query Parquet, CSV, or ORC files **as if they were tables**.

Example:

```
CREATE EXTERNAL TABLE ext_sales (
   id INT,
   amount FLOAT
)
WITH (
   LOCATION = '/sales-data/',
   DATA_SOURCE = MyADLS,
   FILE_FORMAT = ParquetFileFormat
);
```

26. How do you implement security measures for data in transit and at rest in Azure?

✅ **Data at Rest:**

- **Encryption by default** using Azure-managed keys (Storage, SQL, Synapse)
- **Customer-Managed Keys (CMK)** via Azure Key Vault
- **Transparent Data Encryption (TDE)** in SQL

✅ **Data in Transit:**

- **HTTPS and TLS 1.2+** for secure data movement
- Use **Private Endpoints** to keep traffic within Azure network
- **SAS Tokens** and **OAuth tokens** for controlled access

27. How do you use Azure Stream Analytics for real-time data processing?

Azure Stream Analytics (ASA) processes **real-time streaming data** from sources like:

- Azure Event Hubs
- Azure IoT Hub
- Azure Blob Storage (append blobs)
- You can run SQL-like queries on streaming data.

Example:

```
SELECT deviceId, AVG(temperature) AS avgTemp
FROM inputStream TIMESTAMP BY eventTime
GROUP BY deviceId, TumblingWindow(minute, 5)
```

📤 **Output:** Write to SQL DB, Power BI, Blob, ADLS, etc.

Use ASA when you want dashboards or alerts **within seconds/minutes** of event occurrence.

28. What are the different data partitioning strategies in Azure SQL Data Warehouse (Synapse)?

Partitioning helps improve **query performance** and **data management**.

✅ **Common Strategies:**

1. **Hash Partitioning:**
   - Distributes rows using a hash function on a column (e.g., CustomerID)
2. **Range Partitioning:**

- Split data by date ranges, like monthly or yearly partitions

3. **Round-Robin Distribution:**
   - Default method; spreads rows evenly regardless of value
4. **Replicated Tables:**
   - Whole table is copied to all compute nodes (use for small dimension tables)

🧠 Choosing the right strategy depends on your query patterns and data size.

29. How do you monitor and troubleshoot data pipelines in Azure Data Factory (ADF)?

ADF provides built-in **Monitoring** tools:

🔍 **Monitoring Portal**:

- Check pipeline run status: succeeded, failed, in progress
- Inspect activity run logs

📧 **Alerts & Metrics:**

- Set up alerts via Azure Monitor for failure or latency

 Debugging Tools:

- Use **Data Preview** in Mapping Data Flows
- Enable **logging to Log Analytics** or **Storage Accounts**

🔔 Best Practice:

- Use **Retry policies** and **Activity dependencies** to handle failures gracefully.

30. Describe the process of integrating Azure Databricks with Azure Data Lake Storage (ADLS)

🚀 **Steps to integrate Databricks with ADLS Gen2:**

1. **Create Storage Account** (with hierarchical namespace enabled)
2. **Set up Azure Key Vault** (to store credentials securely)
3. **Mount ADLS in Databricks**:

```
configs = {
  "fs.azure.account.auth.type": "OAuth",
  "fs.azure.account.oauth.provider.type": "...",
  "fs.azure.account.oauth2.client.id": "...",
  ...
}

dbutils.fs.mount(
  source = "abfss://<container>@<storageaccount>.dfs.core.windows.net/",
  mount_point = "/mnt/myadls",
  extra_configs = configs
)
```

4. **Read/write data** from /mnt/myadls/

✅ You can also use **ABFS paths directly** in read/write code without mounting.

31. How do you manage and automate data workflows using Azure Logic Apps?

Azure Logic Apps is a no-code/low-code platform to automate workflows like:

- Data movement
- Notification alerts
- File uploads
- Database triggers

📌 **Example Workflow**:

1. Trigger: When a new file is uploaded to Blob
2. Condition: If file name contains “sales”
3. Action: Trigger a stored procedure or pipeline in ADF

📌 Common Connectors:

- SQL Server, Outlook, Blob, ADLS, Service Bus, Salesforce, etc.

32. Explain the concept of Managed Identity in Azure and its use in data engineering

Managed Identity allows Azure services to **authenticate securely with other Azure resources without needing secrets/passwords**.

👤 Two types:

- **System-assigned**: Bound to a single resource
- **User-assigned**: Reusable across multiple resources

🔐 **Use Cases in Data Engineering**:

- ADF accessing Key Vault
- Databricks accessing ADLS
- Synapse accessing Azure SQL

✔ This improves **security, automation, and governance**.

33. How do you ensure data consistency and reliability in distributed systems on Azure?

✔ **Key Practices:**

1. **Idempotent Operations**:
    - Ensure retrying the same operation doesn't lead to duplicates
2. **Exactly-once Delivery** (e.g., Event Hubs + Stream Analytics)
3. **Checkpointing & Watermarking** (e.g., in Databricks structured streaming)
4. **Delta Lake Transactions**:
    - Use **ACID compliance** to manage updates reliably
5. **Retry Policies and Failover**:
    - Handle transient failures with exponential backoff
6. **Monitoring and Alerting**:
    - Use Azure Monitor to detect and react to data issues
7. **Data Quality Checks**:
    - Enforce schema validation, null checks, uniqueness

34. Describe the process of performing ETL operations using Azure Data Factory (ADF)

Azure Data Factory is a **cloud-based ETL tool** that helps you move and transform data across sources and destinations.

🔹 **ETL Workflow in ADF**:

1. **Extract**: Use **Copy Activity** to pull data from sources like SQL, Oracle, SAP, Blob, ADLS, etc.
2. **Transform**:
    - Use **Mapping Data Flows** (visually designed Spark-based transformations)
    - Or trigger **Azure Databricks Notebooks** or **Stored Procedures**
3. **Load**: Write the processed data to a sink like Azure SQL, ADLS, Synapse, etc.

🔄 **Trigger Options**:

- Manual
- Scheduled (time-based)
- Event-based (e.g., blob created)

🔧 **Components**:

- Pipeline (group of activities)
- Linked Service (connection to data)
- Datasets (metadata about data)
- Integration Runtime (compute engine for data movement)

35. How do you use Azure Monitor to track and analyze performance metrics of your data infrastructure?

Azure Monitor helps you **observe, alert, and analyze** the health of your Azure resources.

🔍 **For Data Infrastructure**:

- **Azure Data Factory**: Monitor pipeline runs, trigger executions, activity failures
- **Azure SQL/Synapse**: Track DTU usage, query duration, deadlocks, etc.

- **ADLS**: Monitor storage usage, access patterns

📈 **Key Tools Inside Azure Monitor**:

- **Metrics**: Real-time charts (CPU, memory, latency)
- **Logs**: Deep insights via Log Analytics queries (KQL)
- **Alerts**: Automatic triggers for thresholds (e.g., pipeline failure or high latency)
- **Dashboards**: Custom views to observe performance in one place

36. Explain the role of Azure Key Vault in securing sensitive data

Azure Key Vault is a **secure secrets management** service that stores:

🔐 **What you can store**:

- API keys
- Passwords
- Certificates
- Encryption keys

🎯 **Why it matters**:

- Keeps secrets out of code
- Supports **RBAC** and **Managed Identity** for secure access
- Integrates with ADF, Databricks, Function Apps, Logic Apps, etc.

✅ **Use Cases**:

- ADF securely accessing SQL DB using secrets from Key Vault
- Databricks fetching credentials during runtime

37. How do you handle big data processing using Azure HDInsight?

Azure HDInsight is a **managed cloud service for open-source big data frameworks**.

🔧 **Supported Frameworks**:

- Apache Spark
- Hadoop
- Hive
- HBase
- Kafka

📌 **How to use it**:

1. Choose the right cluster type (e.g., Spark for ETL, Kafka for streaming)
2. Load your data (from Blob, ADLS, etc.)
3. Write your ETL/ML code in PySpark or HiveQL
4. Submit jobs using Ambari, Jupyter, or REST APIs
5. Monitor via Ambari UI or Azure Monitor

🌩️ **Why HDInsight**:

- Enterprise-scale processing
- Supports large volume batch/stream processing
- Integrated with Azure ecosystem

38. What are the best practices for data archiving and retention in Azure?

🗄️ **Archiving = moving less-used data to cheaper storage**, while **retention = how long data is kept**.

🔹 **Best Practices**:

1. **Use ADLS Lifecycle Management**:
    - Automatically move data from **hot → cool → archive** tiers
    - Delete files after a specific retention period
2. **Tag and Classify Data**:
    - Metadata tagging helps apply rules based on importance
3. **Immutable Storage (Write Once, Read Many)**:

- Use this for compliance (e.g., financial/legal data)

4. **Use Azure Blob Archive Tier**:
   - Cheapest option for infrequently accessed data
5. **Document Retention Policies**:
   - Clearly define which datasets are critical, long-term, or disposable

39. How do you implement disaster recovery and backup strategies for data in Azure?

🔒 **Key Components of Disaster Recovery (DR)**:

1. **Geo-Redundancy**:
   - Use **GZRS/RA-GRS** for storage
   - Azure SQL & Synapse have built-in geo-replication
2. **Automated Backups**:
   - Azure SQL: Point-in-time restore, Long-Term Retention (LTR)
   - ADLS: Use **soft delete** and **snapshots**
3. **Cross-Region Replication**:
   - Set up secondary regions for business continuity
4. **Infrastructure as Code**:
   - Use ARM templates or Terraform to quickly redeploy
5. **Runbook/Automation**:
   - Automate failover or restore via Logic Apps or Azure Automation

🔔 **Test your DR strategy periodically** (mock failovers, validation scripts)

SQL

1. How to find the second-highest salary in a table? (Multiple methods)

Assuming table: employees(emp_id, emp_name, salary)

✅ **Method 1: Using DISTINCT + LIMIT / OFFSET (SQL Server: TOP)**

```
SELECT DISTINCT salary
FROM employees
ORDER BY salary DESC
LIMIT 1 OFFSET 1;
```

✅ **Method 2: Using MAX() with a subquery**

```
SELECT MAX(salary)
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);
```

✅ **Method 3: Using DENSE_RANK()**

```
SELECT salary
FROM (
  SELECT salary, DENSE_RANK() OVER (ORDER BY salary DESC) as rnk
  FROM employees
) ranked
WHERE rnk = 2;
```

2. Difference between TRUNCATE and DELETE

Feature	DELETE	TRUNCATE
Can use WHERE clause	✅ Yes	❌ No (removes all rows)
Rollback supported	✅ Yes	✅ Yes (in most modern DBs)
Logging	Fully logged (slower)	Minimal logging (faster)
Resets identity	❌ No	✅ Yes
Triggers	✅ Fires triggers	❌ Does not fire triggers

3. Difference between Stored Procedures and Functions

Feature	Stored Procedure	Function
Return type	Can return 0 or more values	Must return a single value
Use in SELECT	❌ Not allowed	✅ Allowed
DML operations	✅ Can perform INSERT/UPDATE/DELETE	❌ Usually not (varies by DB)
Call syntax	EXEC proc_name	SELECT function_name(...)

4. What are Views and CTEs (Common Table Expressions)?

- **View**: A stored virtual table based on a SQL query.
  Example:
- CREATE VIEW high_salary AS
- SELECT * FROM employees WHERE salary > 100000;
- **CTE**: A temporary result set used within a single SQL query.
  Example:
- WITH dept_avg AS (
- SELECT department, AVG(salary) AS avg_salary
- FROM employees GROUP BY department
-)
- SELECT * FROM dept_avg WHERE avg_salary > 50000;

5. How do aggregation functions differ from analytic functions in SQL?

Feature	Aggregation Functions	Analytic Functions
Return type	Single value per group	Value for each row
GROUP BY needed	✅ Yes	❌ Not required
Examples	SUM(), AVG(), COUNT()	ROW_NUMBER(), LAG(), RANK()

6. SQL Queries

✔ Top 3 earners in each department

```
SELECT * FROM (
  SELECT *, DENSE_RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS rnk
  FROM employees
) sub
WHERE rnk <= 3;
```

✔ Count of duplicates in a table (e.g., duplicate names)

```
SELECT emp_name, COUNT(*)
FROM employees
GROUP BY emp_name
HAVING COUNT(*) > 1;
```

✔ Use LAG() and RANK()

```
SELECT emp_id, emp_name, salary,
      LAG(salary) OVER (PARTITION BY department ORDER BY salary) AS prev_salary,
      RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS rank
FROM employees;
```

✔ Count occurrences of each word in a column (assuming column has strings)

Assume column text_column has space-separated words:

```
-- Depends on DBMS; In SQL Server, you'd use STRING_SPLIT and GROUP BY
WITH words AS (
  SELECT value AS word
  FROM my_table
  CROSS APPLY STRING_SPLIT(text_column, ' ')
)
SELECT word, COUNT(*) AS occurrences
FROM words
GROUP BY word;
```

7. Difference between Star Schema, Snowflake Schema, and 3NF

Feature	Star Schema	Snowflake Schema	3NF (Third Normal Form)
Structure	Central fact table + denormalized dimensions	Fact table + normalized dimensions	Fully normalized (no redundancy)
Query performance	Fast	Slightly slower	Slower (joins needed)
Use case	OLAP/Data Warehouses	OLAP with normalized dimensions	OLTP systems

8. Explain ACID properties in SQL with an example

- **A**tomicity: All steps in a transaction complete or none.
- **C**onsistency: Database remains valid before and after the transaction.
- **I**solation: Concurrent transactions don't interfere.
- **D**urability: Once committed, data is permanent.

 Example: Transferring ₹1000 from Account A to B:

```
BEGIN TRANSACTION;
UPDATE accounts SET balance = balance - 1000 WHERE id = 'A';
UPDATE accounts SET balance = balance + 1000 WHERE id = 'B';
COMMIT;
```

If step 2 fails, step 1 is rolled back.

9. SQL Query to Convert Row-Level Data to Column-Level (Pivot)
Assume table: sales(emp_name, month, revenue)

```
SELECT emp_name,
    SUM(CASE WHEN month = 'Jan' THEN revenue ELSE 0 END) AS Jan,
    SUM(CASE WHEN month = 'Feb' THEN revenue ELSE 0 END) AS Feb,
    SUM(CASE WHEN month = 'Mar' THEN revenue ELSE 0 END) AS Mar
FROM sales
GROUP BY emp_name;
```

✔ For SQL Server: Use PIVOT keyword.

10. How to create and use indexes (clustered vs non-clustered)

- **Clustered Index**: Sorts table physically based on the key.
- CREATE CLUSTERED INDEX idx_emp_salary ON employees(salary);
- **Non-Clustered Index**: Creates a separate structure pointing to rows.
- CREATE NONCLUSTERED INDEX idx_emp_name ON employees(emp_name);

📝 **Tips**:

- Use indexes on columns frequently used in WHERE, JOIN, ORDER BY
- Avoid indexing columns with high updates/inserts

11. Common queries for optimization:
1. Find records that exist in one table but not in another
Using LEFT JOIN:

```
SELECT a.*
FROM table_a a
LEFT JOIN table_b b ON a.id = b.id
WHERE b.id IS NULL;
```

Using NOT IN:

```
SELECT *
FROM table_a
WHERE id NOT IN (SELECT id FROM table_b);
```

Using NOT EXISTS:

```
SELECT *
FROM table_a a
WHERE NOT EXISTS (
   SELECT 1 FROM table_b b WHERE a.id = b.id
);
```

2. Employees earning more than their manager

```
SELECT e.employee_id, e.name, e.salary
FROM employees e
JOIN employees m ON e.manager_id = m.employee_id
WHERE e.salary > m.salary;
```

12. Union vs Union All:

Feature	UNION	UNION ALL
Duplicates	Removes duplicates	Keeps duplicates
Performance	Slower (because of deduplication)	Faster (no deduplication)
Use Case	When unique results are needed	When all data is needed (e.g., logs)

13. Concurrency issues in SQL:
Common issues: **Deadlocks, Dirty Reads, Lost Updates, Non-repeatable Reads**
Solutions:

- Use appropriate **isolation levels** (e.g., READ COMMITTED, SERIALIZABLE).
- Apply **row-level locking** when needed.
- Use **optimistic/pessimistic concurrency control**.
- Avoid long-running transactions.
- Use **retry logic** in applications.

14. Role of Azure SQL Database in cloud-based solutions:
Azure SQL Database is a **fully managed PaaS** offering:

- Auto-patching, backups, scaling.
- High availability with SLA.
- Integration with services like **ADF, Synapse, Key Vault**.
- Built-in **security** and **compliance** features.
- Great for OLTP workloads, APIs, and modern app backends.

15. Setting up ADF pipeline for ETL:

1. **Create linked services** (source and sink).
2. Define **datasets**.
3. Add **Copy Activity** to move data.
4. Use **Data Flows** for transformations (if needed).
5. Add control flow using **ForEach**, **If**, etc.
6. Publish pipeline and trigger it via **schedule/event/manual**.

16. Best practices for Azure SQL Database security:

- Use **Azure Active Directory (AAD)** authentication.
- Enable **Transparent Data Encryption (TDE)**.
- Use **Always Encrypted** for sensitive columns.
- Secure secrets with **Azure Key Vault**.
- Apply **firewall rules and VNET integration**.
- Use **Auditing & Threat Detection** features.

17. Data partitioning in Azure SQL Data Warehouse:

- Use **table partitioning** based on a key (e.g., date).
- Improves query performance and manageability.
- Use **round-robin, hash, or replicated** distribution.
- Best practices:
    - Use **hash distribution** for large fact tables.
    - Replicate small dimension tables.
    - Avoid skewed partitions.

18. Azure Cosmos DB vs Traditional SQL:

Feature	Azure Cosmos DB	Traditional SQL DB
Data Model	NoSQL (key-value, doc, graph, etc.)	Relational
Performance	Low-latency, global distribution	Not built for global scale
Scalability	Elastic, horizontal	Limited vertical scaling
Use Case	IoT, social, gaming, real-time	OLTP, transactional apps

19. Integrating Azure SQL with other services:

- **ADF**: Move/transform data.
- **Power BI**: Reporting and dashboards.
- **Logic Apps**: Automate workflows.
- **Azure Functions**: Custom triggers and automation.
- **Key Vault**: Secure connection strings.

- **Synapse**: Analytics and data warehouse offloading.

20. Azure Data Lake + SQL-based systems integration:

- Use **PolyBase** in Synapse/SQL DW to query files in ADLS.
- Use **ADF** to copy/move data from ADLS to SQL.
- Use **Spark (Databricks)** to clean/prepare data, then write to SQL.
- Integration is seamless through **linked services and managed identities**.

21. Azure Logic Apps for SQL automation:

- Triggered by events (new row, file, HTTP).
- Connectors available for **SQL Server, Azure SQL, ADLS**, etc.
- Actions: Insert, update, delete records.
- Used for alerts, syncs, automated maintenance, etc.

22. Migrating on-prem SQL to Azure SQL:

Steps:

1. Use **Data Migration Assistant (DMA)** for assessment.
2. Choose target: Azure SQL DB, Managed Instance, or SQL VM.
3. Use **Azure Database Migration Service (DMS)** for online/offline migration.
4. Reconfigure connection strings.
5. Test and validate.

23. Azure Data Lake Analytics + U-SQL:

- U-SQL = SQL + C# for big data processing.
- Used with ADLA (now deprecated in favor of Spark).
- Works on unstructured/semi-structured data.
- Great for:
    - Data cleansing
    - Extract-transform-load (ETL)
    - Aggregations and joins on large datasets

24. Differences between Azure SQL Database and Azure SQL Managed Instance

Feature	Azure SQL Database	Azure SQL Managed Instance
Deployment Type	Single DB or Elastic Pool	Instance-level (multiple DBs like on-prem SQL Server)
Feature Compatibility	Limited (no SQL Agent, CLR, etc.)	Near-full compatibility with on-prem SQL Server
Use Case	Modern cloud apps needing high availability	Lift-and-shift on-prem workloads with minimal changes
VNET Support	Limited (private endpoint)	Full VNET support
Cross-DB Queries	Not supported	Supported

25. Ensuring High Availability & Disaster Recovery in Azure SQL DB

- **Built-in HA** via **Premium/Business Critical** tiers.
- **Geo-replication** (active-passive): creates readable secondary in a different region.
- **Auto-failover groups**: automatic disaster recovery with failover policy.
- **Backups**:
    - Automatic backups (7–35 days retention).
    - Long-term retention (LTR) for years.
- **Zone-redundant configuration** for higher resiliency.

26. Common challenges in managing large-scale SQL databases in Azure

- **Performance tuning**: query slowness, index maintenance.
- **Cost management**: especially with provisioned resources.
- **Scaling**: vertical scaling has limits.
- **Concurrency issues** with high user volume.
- **Monitoring complexity** across regions/replicas.
- **Automation**: backups, index rebuilds, and patching need oversight.

27. Optimizing Query Performance in Azure SQL Database

- Use **Query Performance Insight** for slow queries.
- Implement **proper indexing** (clustered/non-clustered, filtered).
- Analyze execution plans for **missing indexes, scans vs seeks**.
- Use **Query Store** for performance history and regressions.
- Tune parameters and avoid **parameter sniffing**.
- Archive/historicize old data to reduce table size.

28. Setting up and managing an Azure Synapse Analytics workspace

1. **Create Synapse Workspace** via Azure portal.
2. Link **Azure Data Lake Storage Gen2** account as default storage.
3. Enable **Managed Identity** for secure resource access.
4. Use **Synapse Studio** for:
    - Notebooks (Spark),
    - SQL scripts (dedicated & serverless),
    - Data flows, pipelines, monitoring.
5. Integrate with **Power BI, ADF, Azure ML**, etc.
6. Monitor workloads and query plans via **Synapse Studio > Monitor tab**.

29. Monitoring and Troubleshooting Azure SQL Database

- Use **Azure Monitor + Log Analytics** for telemetry.
- Enable **SQL Insights** for deep diagnostics.
- Monitor via:
    - **Query Store**
    - **Performance metrics** (DTU, CPU, Memory, IO, blocking sessions)
    - **Intelligent Insights** (auto-detect anomalies)
- Set up **alerts** for key metrics (e.g., CPU > 90%).

30. Azure Active Directory (AAD) for SQL DB Authentication

- Enables **centralized identity management**.
- Supports **MFA**, conditional access, and role-based access.
- Steps:
    1. Add AAD Admin to SQL server.
    2. Create contained database users mapped to AAD identities/groups.
    3. Connect using AAD token (SSMS, Azure Data Studio, or app).
- Benefits: **No password storage**, better **security & auditing**.

31. Implementing Data Encryption in Azure SQL Database

- **At Rest**:
    - **Transparent Data Encryption (TDE)** – enabled by default.
    - Integration with **Azure Key Vault** for BYOK (Bring Your Own Key).
- **In Transit**:
    - Enforced via **SSL/TLS** (automatically handled by Azure).
- **Column-Level Encryption**:
    - **Always Encrypted** – encrypts sensitive columns (e.g., SSN).

32. Using Azure SQL Data Sync for replication

- Allows **bi-directional data sync** between Azure SQL and:
  - Other Azure SQL DBs
  - SQL Server (on-premises)
- Components:
  - **Sync Group**
  - **Hub Database** (central)
  - **Member Databases** (can be multiple)
- Useful for **hybrid sync, geo-distributed apps**, or local caching.
- Managed via Azure Portal.

33. Key considerations for designing scalable data architecture in Azure

- **Data Partitioning**: Improve performance and parallelism.
- **Decouple compute and storage** (e.g., using ADLS + Synapse).
- Use **Data Lakes** for raw/semi-structured data.
- **Choose the right storage tier** (Hot, Cool, Archive).
- Use **serverless options** for ad-hoc or infrequent queries.
- Design for **modularity** and **reuse** (ADF pipelines, notebooks).
- Enable **monitoring, logging, and alerting** from the start.
- Leverage **Autoscaling** and **Pay-as-you-go** services to control cost.

SQL Important coding ques

1. Find the second-highest salary in a table

```
SELECT MAX(salary) AS SecondHighestSalary
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);
```

Alternative using DENSE_RANK:

```
SELECT salary
FROM (
  SELECT salary, DENSE_RANK() OVER (ORDER BY salary DESC) as rnk
  FROM employees
) AS ranked
WHERE rnk = 2;
```

2. Top 3 earners in each department

```
SELECT *
FROM (
```

```
  SELECT *, DENSE_RANK() OVER (PARTITION BY department ORDER BY salary DESC) as rnk
  FROM employees
) as ranked
WHERE rnk <= 3;
```

3. Count of duplicates in a table

```
SELECT column_name, COUNT(*) as count
FROM table_name
GROUP BY column_name
HAVING COUNT(*) > 1;
```

4. Use LAG and RANK functions

```
SELECT employee_id, salary,
      LAG(salary, 1) OVER (ORDER BY salary) as PrevSalary,
      RANK() OVER (ORDER BY salary DESC) as SalaryRank
FROM employees;
```

5. Count occurrences of each word in a table column

Assume the column is comments:

```
WITH Words AS (
  SELECT value AS word
  FROM table_name
  CROSS APPLY STRING_SPLIT(comments, ' ')
)
SELECT word, COUNT(*) as count
FROM Words
GROUP BY word;
```

6. Convert row-level data to column-level using pivot

```
SELECT *
FROM (
  SELECT department, month, salary
  FROM salaries
) src
PIVOT (
  SUM(salary) FOR month IN ([Jan], [Feb], [Mar])
) AS pvt;
```

7. Find records in one table but not in another

```
SELECT * FROM table1
WHERE id NOT IN (SELECT id FROM table2);
```

8. Find employees earning more than their manager

```
SELECT e.employee_id, e.salary
FROM employees e
JOIN employees m ON e.manager_id = m.employee_id
WHERE e.salary > m.salary;
```

9. Cumulative sum of a column

```
SELECT employee_id, department, salary,
      SUM(salary) OVER (PARTITION BY department ORDER BY employee_id) AS cumulative_salary
FROM employees;
```

10. Nth highest salary

```
SELECT DISTINCT salary
FROM (
  SELECT salary, DENSE_RANK() OVER (ORDER BY salary DESC) as rnk
  FROM employees
) ranked
WHERE rnk = N;  -- Replace N with the desired rank
```

11. Employees who joined in the last 6 months

```
SELECT * FROM employees
WHERE joining_date >= DATEADD(MONTH, -6, GETDATE());
```

12. Department-wise highest salary

```
SELECT department, MAX(salary) as max_salary
FROM employees
GROUP BY department;
```

13. Join two tables and display specific columns

```
SELECT e.employee_id, e.name, d.department_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id;
```

14. Find gaps in a sequence of numbers

```
SELECT number + 1 AS missing
FROM numbers
WHERE number + 1 NOT IN (SELECT number FROM numbers);
```

15. Remove duplicate rows

```
DELETE FROM employees
WHERE id NOT IN (
  SELECT MIN(id)
  FROM employees
  GROUP BY name, salary, department
);
```

16. Average salary by department

```
SELECT department, AVG(salary) as avg_salary
FROM employees
GROUP BY department;
```

17. Median salary of employees

SQL Server (using PERCENTILE_CONT):

```
SELECT PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY salary) OVER () as median_salary
FROM employees;
```

18. Rank employees based on performance

```
SELECT employee_id, performance_score,
     RANK() OVER (ORDER BY performance_score DESC) as rank
FROM employees;
```

19. Employees without a manager

```
SELECT * FROM employees
WHERE manager_id IS NULL;
```

20. Create new table with results of a complex query

```
SELECT department, AVG(salary) as avg_salary
INTO DepartmentSalary
FROM employees
GROUP BY department;
```

Azure Databricks

🔧 **1. What is the difference between a job cluster and an interactive cluster?**

- **Job Cluster:** Temporary, created for a job and terminated after. Optimized for scheduled/automated workflows.
- **Interactive Cluster:** Always running, used for development/debugging and ad hoc analysis via notebooks.

🏗️ **2. Explain Databricks Lakehouse architecture (Bronze, Silver, Gold layers)**

- **Bronze:** Raw ingestion from source systems (minimal transformation).
- **Silver:** Cleaned, validated, and enriched data.
- **Gold:** Aggregated, business-level curated data for reporting/analytics.

📎 **3. How to copy files from Blob Storage to Databricks?**

You can use:

```
dbutils.fs.cp("wasbs://<container>@<account>.blob.core.windows.net/<path>",
        "dbfs:/mnt/<mount-name>/<path>",
        recurse=True)
```

Or mount the Blob Storage first using dbutils.fs.mount.

📓 4. How to create and deploy notebooks in Databricks?

- Create: In the Databricks workspace → New → Notebook.
- Deploy: Use **Jobs** or **Workflows** to schedule and trigger execution. You can also integrate with CI/CD.

📚 5. What is Unity Catalog in Databricks?

- A centralized governance layer for managing data access, lineage, auditing, and discovery across all Databricks workspaces.
- Works with fine-grained access control at table, column, and row level.

🔁 6. How to run one notebook from another?

```
# Using %run (for shared variables/functions)
%run "./notebook_path"

# Using dbutils (returns value, runs in new context)
dbutils.notebook.run("notebook_path", timeout_seconds=60, arguments={"key":"value"})
```

🔌 7. How to connect ADLS to Databricks?

Use either mounting or direct path:

```
# Direct access
df = spark.read.format("parquet").load("abfss://<container>@<storage-account>.dfs.core.windows.net/<path>")

# Or mount using service principal / managed identity
dbutils.fs.mount(...)
```

📜 8. What are Delta logs, and how to track data versioning in Delta tables?

- Delta uses _delta_log/ directory to store JSON files (transaction logs).
- Use:

```
DESCRIBE HISTORY delta.`/path/to/table`
```

To see version history, timestamp, operation type, etc.

🔄 9. How to implement SCD Type 1 and Type 2 using PySpark in Databricks?

- **SCD Type 1:** Overwrite old records.
- **SCD Type 2:** Maintain history by using is_current, start_date, end_date columns.

Delta Lake makes it easier using MERGE INTO:

```
MERGE INTO target t
USING source s
ON t.id = s.id
WHEN MATCHED AND (t.col1 != s.col1) THEN UPDATE SET ...
WHEN NOT MATCHED THEN INSERT ...
```

⚙️ 10. How do autoscaling clusters work in Databricks?

- Automatically increase/decrease workers based on workload.
- You define min and max workers in cluster settings.
- Helps optimize cost and performance.

🚀 11. Databricks optimization techniques

- Use **Delta Lake** for ACID transactions and fast reads.
- Partition large tables.
- Use **Z-ordering** for efficient filtering.
- Cache frequently accessed data.

- Use **broadcast joins** for small lookup tables.

📊 **12. How to monitor Databricks jobs?**
- Use the **Jobs UI** for job status, run history, logs.
- View logs via **Spark UI**, **Driver/Worker logs**.
- Use **Cluster Metrics tab** for memory, CPU usage.

🔄 **13. Difference between caching and persisting**
- **cache()** → Stores in-memory with default storage level (MEMORY_AND_DISK).
- **persist()** → You can specify storage level, e.g., memory only, disk only, etc.

🔐 **14. How to give permission to specific notebooks or clusters to users?**
- Go to notebook/cluster → Permissions tab → Add users/groups → Assign role (Can View, Can Run, Can Edit).

✔️ **15. How to implement data validation and quality checks in Databricks?**
- Use **assertions** or **expectations** (with Deequ or Great Expectations).
- Example:

```
assert df.filter("salary < 0").count() == 0, "Negative salary found!"
```

⏱️ **16. How to manage and automate ETL workflows using Databricks Workflows?**
- Use **Workflows** (Jobs UI) → Create tasks → Chain notebooks/scripts → Define schedule, retries, parameters.

🔐 **17. Best practices for securing data in Databricks**
- Use **Unity Catalog** + **Access Control Lists (ACLs)**.
- Use **Credential Passthrough** or **Managed Identity**.
- Enable **IP access control**, **secure cluster connectivity**.
- Encrypt data at rest (via Azure Storage) and in transit (TLS).

🔄 **18. Integrating Databricks with Azure DevOps for CI/CD pipelines**
- Store notebooks in **Git repo** (Azure Repos).
- Use **Databricks CLI / REST API** to deploy via pipelines.
- Example CI/CD tools: Azure Pipelines + Databricks CLI + Terraform.

🔁 **19. How to use Databricks Delta Live Tables (DLT) for continuous processing?**
- DLT is a declarative ETL framework.
- Use SQL or Python to define **LIVE TABLES**.
- It handles:
    - Data quality
    - Incremental loads
    - Lineage
    - Monitoring

Example:

```
CREATE LIVE TABLE bronze AS SELECT * FROM cloud_files("path", "json")
```

Azure Data Factory

1. How to implement incremental load in ADF?
Use watermark columns (e.g., LastModifiedDate) and store the last loaded value in a variable or file.
In the next run, filter data using a query like:
SELECT * FROM table WHERE LastModifiedDate > @lastLoadedTime
Use a Lookup activity to get the last value, then set it via a variable or parameter in the pipeline.

2. How do full loads differ from incremental loads in ADF?

- **Full Load**: Loads the entire dataset every time (can be slow and resource-heavy).
- **Incremental Load**: Loads only new or changed data based on a watermark (efficient and scalable).

3. What are the types of Integration Runtimes (IR) in ADF?

1. **Azure IR**: For cloud data movement and transformation.
2. **Self-hosted IR**: For on-premises or VNET-based data sources.
3. **Azure-SSIS IR**: To run SSIS packages in Azure.

4. Explain the process of copying large files (e.g., 3TB) from on-premises to Azure in ADF.

- Use **Self-hosted IR**.
- Enable **parallelism** via data partitioning.
- Use **binary copy** for non-transformational moves.
- Consider **staging in Azure Blob Storage** first.

5. How to handle error handling in ADF using retry, try-catch blocks, and failover mechanisms?

- **Retry**: Configure retries in activity settings.
- **Try-Catch**: Use **'If Condition'**, **'Switch'**, and **'Until'** activities with proper success/failure dependencies.

- **Failover**: Route failures to a different branch (e.g., send email, log error, retry with backup).

6. What is the binary copy method, and when is it used?
Used when copying **files as-is**, without parsing. Ideal for **images, videos, ZIP files**, etc. It improves performance and avoids unnecessary processing.

7. How to perform parallel copies in ADF using partitioning?
Enable **"Parallel Copy"** in the **Copy Activity**, and partition the source using:
- **Dynamic Range Partition** (e.g., based on ID or Date)
- **Fixed Range Partition**
- **Hash Partition**

8. How do you handle complex ADF pipelines? Explain challenges faced.
- Use **modular pipelines**, pass parameters.
- Use **metadata-driven design**.
- Challenges: Managing dependencies, debugging failures, handling schema drift.

9. What are the activities in ADF (e.g., Copy Activity, Notebook Activity)?
- **Copy Data Activity**
- **Data Flow**
- **Execute Pipeline**
- **Lookup**
- **Web/REST API**
- **Execute Notebook**
- **Set Variable, If Condition, ForEach**

10. How to pass parameters from ADF to Databricks notebooks?
In **Notebook Activity**, use the baseParameters property:
```
"baseParameters": {
  "input_date": "@pipeline().parameters.date"
}
```
Then use dbutils.widgets.get("input_date") in the notebook.

11. How to connect to Salesforce using ADF securely?
- Use **Salesforce connector** with OAuth2.
- Store secrets in **Azure Key Vault**.
- Use **Linked Services** with credential references.

12. Where to store sensitive information like passwords in ADF (e.g., Azure Key Vault)?
Store in **Azure Key Vault**, and reference secrets in **Linked Services** using the Key Vault integration.

13. How to copy all tables from one source to the target using metadata-driven pipelines?
- Create a metadata table with source/target table names.
- Use **Lookup** + **ForEach** to iterate and pass values to a **Copy Activity** dynamically.

14. Trigger types in ADF (Schedule, Tumbling Window, etc.)
- **Schedule Trigger** – runs at fixed time.
- **Tumbling Window** – handles retry and backfill (ideal for incremental).
- **Event-based** – triggers on file drop.
- **Manual/On-demand** – triggered manually or via REST API.

15. How to implement error retry policies in ADF pipelines?
- Set retry count and interval in the activity settings.
- Use **on-failure** dependency path for custom error handling.

16. How to implement data validation and data quality checks in ADF?

- Use **Data Flows** with checks like null count, duplicates.
- Use **Stored Procedures** or **Notebooks**.
- Fail pipeline using **assertions** or **If conditions**.

17. How do you optimize the performance of ADF pipelines?

- Use **parallelism and partitioning**.
- **Filter early** in source queries.
- Minimize **data movement**.
- Use **staging** for transformations.

18. Describe the process of integrating ADF with Azure Synapse Analytics.

- Use **Linked Service** to connect to Synapse.
- Load data via **Copy Activity** or **Stored Proc activity**.
- Trigger **Synapse Notebooks or SQL Scripts** for processing.

19. How do you handle schema drift in ADF?

- Enable **"Allow Schema Drift"** in **Mapping Data Flows**.
- Use **Auto Mapping**, and **dynamic column handling**.
- Maintain **metadata table** to track schema changes.

20. Explain how to use ADF with Azure Databricks for complex transformations.

- Use **Notebook Activity** in ADF.
- Pass parameters to Databricks.
- Databricks handles transformation logic; output can be written to ADLS or Synapse.

21. How do you implement data masking in ADF for sensitive data?

- Mask in **Mapping Data Flows** using expressions.
- substring(col, 1, 2) + '***'
- Mask using SQL views or Databricks logic before loading.

22. Describe the steps to migrate SSIS packages to ADF.

- Create **Azure-SSIS Integration Runtime**.
- Deploy SSIS packages to **Azure SSISDB** using SSMS or SSIS projects.
- Execute packages using **Execute SSIS Package** activity in ADF.

23. How do you monitor and troubleshoot ADF pipeline failures?

- Use **Monitor tab** in ADF Studio.
- Enable **Activity-level logging**.
- Integrate with **Log Analytics**, **Alerts**, and **Azure Monitor** for deeper insights.

24. Explain the role of ADF in a hybrid data integration scenario.

ADF supports **on-prem to cloud** or **cloud-to-cloud** data movement using **Self-hosted IR**. It's great for hybrid environments where some systems remain on-prem.

Data Lake Storage (ADLS)

✔ 1. Difference between Blob Storage and ADLS

Feature	Blob Storage	Azure Data Lake Storage (Gen2)
Purpose	General object storage	Big data analytics
Hierarchical namespace	✖ No	✔ Yes (file system-like structure)
Performance for analytics	Moderate	Optimized for big data
Security & Access Control	RBAC, SAS	RBAC + POSIX ACLs
Integration	Suitable for apps	Suited for Hadoop/Spark/ADF

✔ 2. Why is mounting preferred over access keys in Databricks?

- **Mounting** using dbutils.fs.mount:
  - Secure via **Azure Key Vault**.
  - Simplifies access via /mnt path.
  - Long-term use across notebooks.
- **Access Keys** are hard-coded and risky if exposed.

✔ 3. Ensuring security when connecting Databricks with ADLS

- Use **Service Principals with RBAC**.
- Store secrets in **Azure Key Vault**.
- Use **OAuth 2.0 or Managed Identity** for secure token-based access.
- Avoid hardcoding credentials.

✔ 4. Copy all file formats (CSV, Parquet, Excel) using ADF

- Use **Copy Activity** with:
  - Wildcard paths like *.csv, *.parquet.
  - Source dataset with flexible formats.
  - Use **Binary copy** for unstructured formats if needed.
  - Set format property dynamically if part of metadata-driven framework.

✔ 5. Track file names in ADF

- In **Copy Activity**, use Source -> Additional columns:
    - Set column name = "FileName"
    - Value = @item().name
- This captures the file name into the sink table.

✔ 6. Optimize data retrieval from ADLS in Spark

- Partitioned folders (e.g., year=2024/month=01)
- Use **filter pushdown** (spark.read.parquet(...).filter(...))
- Avoid small files ("small files problem")
- Enable **caching** for repeated reads.

✔ 7. ADLS Security Features

- **Azure RBAC**
- **POSIX-style ACLs** (Access Control Lists)
- **Azure Key Vault** integration
- **Data encryption** (at rest & in transit)

✔ 8. Versioning in ADLS

- ADLS Gen2 doesn't have built-in versioning.
- Use **Delta Lake on top of ADLS** to manage data versioning with Delta logs.

✔ 9. Hierarchical Namespace

- Makes ADLS Gen2 behave like a file system.
- Enables:
    - Directory-level operations (rename, delete)
    - ACLs at folder/file level
    - Better integration with analytics engines

✔ 10. Data Lifecycle Policies

- Configure **Lifecycle Management**:
    - Delete after X days
    - Move to cooler tier (e.g., Hot → Cool → Archive)
- Set via Azure Portal or ARM templates.

✔ 11. Differences: ADLS Gen1 vs Gen2

Feature	ADLS Gen1	ADLS Gen2
Storage base	Proprietary	Blob Storage
Integration	Limited	Broad (ADF, Databricks, Synapse)
Cost	Higher	Cheaper (Blob pricing)
Hierarchical Namespace	Always enabled	Optional
Access Control	ACL only	RBAC + ACL

✔ 12. Monitor & Troubleshoot ADLS

- **Azure Monitor**
- **Storage Analytics Logs**
- **Azure Activity Logs**
- Integration with **Log Analytics / Azure Sentinel**

✔ 13. Large-scale data ingestion

- Use:
    - **Azure Data Factory with parallel copy**
    - **Databricks Autoloader**

    - **Azure Event Grid** (for triggers)
    - Consider **batching** and **partitioning**

✔ 14. Disaster Recovery Setup

- Enable **Geo-redundant storage (GRS)**
- Replication: LRS → ZRS → GRS → RA-GRS
- Regular **backups to another region**
- Test **failover and restore processes**

✔ 15. ADLS + Synapse Integration

- Create **Linked Service** in Synapse.
- Use **serverless SQL pools** to query files directly.
- Load ADLS data into dedicated SQL pools via COPY.

✔ 16. Data Quality & Validation

- Implement in:
    - **ADF Data Flow** (Conditional Split, Derived Column)
    - **Databricks** (assertions, expectations)
- Track failed rows to a quarantine folder/table.

✔ 17. Setting up ACLs in ADLS

- Use:
    - Azure Portal
    - Azure CLI: az storage fs access
    - PowerShell or REST API
- Set permissions at folder/file level.

✔ 18. Encryption at Rest & In Transit

- **At Rest**: Automatically with Microsoft or customer-managed keys.
- **In Transit**: TLS/SSL encryption.
- Optional: **Double encryption** for sensitive workloads.

✔ 19. ADLS + Databricks Integration

- Use:
    - **Mounting via dbutils**
    - **Direct access with OAuth/Service Principal**
    - **Delta Lake** format for efficient processing

✔ 20. Best Practices for Storage Costs

- Use **Cool/Archive tiers** for infrequently accessed data.
- Enable **lifecycle policies**.
- Delete obsolete files regularly.
- Avoid storing small files — merge them.

✔ 21. Schema Evolution Handling

Great question! Handling schema evolution in ADLS (Azure Data Lake Storage) depends on the tools you're using on top of ADLS, since ADLS itself is just storage—it doesn't enforce or manage schema. Schema evolution is actually managed by processing engines like Delta Lake (Databricks), ADF, Synapse, or Spark.

✔ Approach 1: Using Delta Lake on ADLS (Databricks or Synapse)

Delta Lake handles schema evolution natively, making it the most efficient way to manage evolving

data structures.

- How to enable schema evolution:

```
df.write \
 .format("delta") \
 .option("mergeSchema", "true") \
 .mode("overwrite") \
 .save("abfss://your-container@your-storage-account.dfs.core.windows.net/path")
```

- Benefits:
  - Adds new columns automatically.
  - Keeps a history of schema versions (data versioning).
  - Supports time travel to roll back to previous versions.

✔ Approach 2: Using Azure Data Factory (ADF)

ADF handles schema drift and evolution using Mapping Data Flows.

- Steps:
  1. In the source and sink, enable "Allow schema drift".
  2. Enable "Auto Mapping" or manually map fields.
  3. Use Derived Columns to handle transformations for new/changed columns.

⚠ ADF does not support adding new columns to SQL sinks automatically. You may need to modify the schema manually or use stored procedures.

✔ Approach 3: Using Azure Synapse Analytics

If querying files directly from ADLS via serverless SQL pools:

- Schema is inferred from the files.
- To evolve schema:
  - Update your CREATE EXTERNAL TABLE or use OPENROWSET with schema declaration.

✔ Best Practices for Schema Evolution in ADLS

Practice	Why It Helps
Use Delta Lake format	Supports schema merging, versioning
Enable "mergeSchema" in Databricks	Allows new columns to be added
Use Parquet or Delta instead of CSV	Self-describing formats support schema
Maintain schema registry (e.g., Azure Purview)	Tracks data model changes across zones
Track schema changes in dev/test before prod	Avoids breaking downstream systems

✔ 22. Explain the process of auditing and logging access to data in ADLS.

- Enable:
  - **Azure Diagnostics**
  - **Activity Logs**
  - **Storage Analytics**
  - Monitor with **Azure Monitor + Log Analytics**

Auditing and logging access to data in **Azure Data Lake Storage (ADLS)**—especially Gen2—is **crucial** for maintaining data security, tracking usage, and meeting compliance requirements (like GDPR, HIPAA, etc.).

Here's how you can **audit and log access** to ADLS Gen2 step-by-step:

✔ 1. Enable Azure Storage Logging and Monitoring
ADLS Gen2 is built on top of Azure Blob Storage, so you use **Azure Monitor**, **Diagnostic Settings**, and **Azure Activity Logs** to audit access.

- **Step-by-step: Enable Diagnostic Logs**
  1. **Go to the storage account** in the Azure Portal.
  2. Click on **"Diagnostics settings"** under the **Monitoring** section.
  3. Click **"+ Add diagnostic setting"**.
  4. Choose the **log types** to send (important ones for auditing):
     - Read (successful read access to files)
     - Write (file uploads, changes)
     - Delete (file deletions)
     - List (directory/file listing)
     - AllMetrics (performance data)
  5. Choose a **destination** for logs:
     - **Log Analytics Workspace** (recommended for query & alerting)
     - **Storage Account** (for long-term storage)
     - **Event Hub** (for real-time event processing)

- **Log Data Examples**

Sample log entry from Log Analytics might show:

```
{
  "callerIpAddress": "203.0.113.1",
  "operationName": "ReadFile",
  "statusCode": "200",
  "authenticationType": "AAD",
  "resourceId": "/subscriptions/.../storageAccounts/yourADLS"
}
```

✔ 2. Enable Azure Activity Logs (for management plane)

- These logs track **who did what** on the **Azure Resource itself** (e.g., permissions changed, blob containers created).
- Go to **Monitor > Activity Log**.
- Filter by **resource type = Storage accounts**.

✔ 3. Use Access Control (IAM) and Azure RBAC

- Audit **who has access** via IAM roles like Storage Blob Data Reader, Contributor, etc.
- Keep least privilege principle.
- Use **Azure Policy** to restrict public access and enforce rules.

✔ 4. Use Azure Purview or Microsoft Defender for Cloud (optional)

- **Azure Purview**: For **data cataloging, classification, and lineage tracking**.
- **Defender for Storage**: For **threat detection** (e.g., unusual access patterns, malware detection).

✔ 5. Query Logs with Log Analytics (KQL)

Example Kusto query to track file reads:

```
StorageBlobLogs
| where OperationName == "GetBlob"
```

| summarize count() by CallerIpAddress, UserPrincipalName, Resource, TimeGenerated

✔ 6. Set up Alerts

Set up alerts for:

- High volume of deletes.
- Access from unknown IPs.
- Unauthorized access attempts.

🔐 Summary: What Can You Audit?

Audit Type	Tool
File access (read/write/delete)	Azure Monitor (Storage Logging)
Permission/config changes	Activity Logs
User actions	Log Analytics / Azure AD Sign-ins
Data sensitivity & classification	Azure Purview
Threat detection	Microsoft Defender for Storage

PYTHON

Code-Based Questions

1. **Generate Prime Numbers**:

```
def generate_primes(n):
    primes = []
    for num in range(2, n + 1):
        if all(num % i != 0 for i in range(2, int(num ** 0.5) + 1)):
            primes.append(num)
    return primes
```

2. **Check if a String is a Palindrome**:

```
def is_palindrome(s):
    s = s.lower().replace(" ", "")
    return s == s[::-1]
```

3. **Replace Vowels with Spaces**:

```
def replace_vowels(text):
    return ''.join(' ' if ch.lower() in 'aeiou' else ch for ch in text)
```

4. **Count Word Occurrences**:

```
from collections import Counter

def count_words(text):
    words = text.lower().split()
    return Counter(words)
```

5. **Find Number That Appears Consecutively 3 Times**:

```
def find_consecutive_triplet(nums):
    for i in range(len(nums) - 2):
        if nums[i] == nums[i+1] == nums[i+2]:
            return nums[i]
    return None
```

6. **Split Name into Firstname and Lastname**:

```
def split_name(full_name):
```

```
    first, last = full_name.strip().split(' ', 1)
    return first, last
```

7. **Generate Fibonacci Numbers**:

```
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        print(a, end=" ")
        a, b = b, a + b
```

8. **Find Duplicates and Count Occurrences**:

```
from collections import Counter

def find_duplicates(lst):
    counts = Counter(lst)
    return {k: v for k, v in counts.items() if v > 1}
```

1. How do you use dictionaries in Python to store key-value pairs efficiently?

Dictionaries in Python are hash tables that allow **O(1)** average-time complexity for lookups, insertions, and deletions. They store data in a {key: value} format.

Example:

```
employee = {
    'name': 'Alice',
    'department': 'HR',
    'salary': 60000
}
print(employee['name'])  # Output: Alice
```

Efficiency Tips:

- Use immutable keys (like strings or numbers).
- Avoid nesting if you don't need it (for readability).
- Use defaultdict or Counter from collections for counting operations.

2. Write Python code to split a name column into firstname and lastname.

Assuming you have a list of full names:

```
names = ['John Doe', 'Jane Smith']
split_names = [name.split(' ', 1) for name in names]
for first, last in split_names:
    print(f"First Name: {first}, Last Name: {last}")
```

With pandas:

```
import pandas as pd

df = pd.DataFrame({'full_name': ['John Doe', 'Jane Smith']})
df[['first_name', 'last_name']] = df['full_name'].str.split(' ', 1, expand=True)
```

3. Generate Fibonacci numbers.

```
def generate_fibonacci(n):
    fib = [0, 1]
    for i in range(2, n):
        fib.append(fib[-1] + fib[-2])
    return fib[:n]
```

```
print(generate_fibonacci(10))
```

4. Identify duplicates in a list and count their occurrences.

```
from collections import Counter

items = ['apple', 'banana', 'apple', 'orange', 'banana']
counts = Counter(items)
duplicates = {item: count for item, count in counts.items() if count > 1}
print(duplicates)
```

5. Explain the difference between lists, tuples, and sets in Python.

Type	Mutable	Ordered	Allows Duplicates	Use Case
List	Yes	Yes	Yes	General-purpose collections
Tuple	No	Yes	Yes	Fixed-size, immutable collections
Set	Yes	No	No	Unique items, fast membership

6. How do you handle missing or null values in a dataset using Python?

Using pandas:

```
import pandas as pd

df = pd.DataFrame({'A': [1, 2, None, 4], 'B': [None, 2, 3, 4]})
df.fillna(0, inplace=True)     # Replace with 0
df.dropna(inplace=True)        # Drop rows with nulls
```

You can also use isnull() or notnull() to filter.

7. Describe the process of data serialization and deserialization in Python.

Serialization = Convert Python object → byte stream (store/send). Deserialization = byte stream → Python object.

Using JSON:

```
import json

data = {'name': 'Alice', 'age': 30}
json_string = json.dumps(data)
restored_data = json.loads(json_string)
```

Using Pickle (for non-JSON types):

```
import pickle

with open('data.pkl', 'wb') as f:
    pickle.dump(data, f)

with open('data.pkl', 'rb') as f:
    loaded_data = pickle.load(f)
```

8. How do you read and write data to and from a database using Python?

Use pyodbc, sqlalchemy, or psycopg2 depending on the DB.

Using sqlalchemy and pandas:

```
from sqlalchemy import create_engine
import pandas as pd

engine = create_engine('mssql+pyodbc://username:password@dsn')
df = pd.read_sql('SELECT * FROM employees', engine)
```

df.to_sql('employees_backup', engine, if_exists='replace', index=False)

9. Explain the concept of list comprehensions and provide an example.
List comprehensions are a concise way to create lists.
Example:
squares = [x**2 for x in range(10)]
With condition:
even_squares = [x**2 for x in range(10) if x % 2 == 0]

10. How do you optimize Python code for better performance?
- Use built-in functions and libraries (e.g., sum(), map())
- Use generators instead of lists for large data
- Avoid global variables
- Use list comprehensions over loops
- Use efficient data structures (set, dict)
- Profile with cProfile or timeit

11. Describe the use of context managers in Python.
Context managers manage setup and teardown (like opening/closing files).
```
with open('file.txt', 'r') as file:
    content = file.read()
# File automatically closed here
```
You can create custom context managers using contextlib or __enter__/__exit__.

12. How do you handle exceptions and errors in Python?
```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
except Exception as e:
    print("Error:", e)
finally:
    print("This always runs.")
```
Use raise to manually throw exceptions.

13. Explain the role of pandas library in data manipulation.
Pandas is a powerful library for handling structured data.
Key Features:
- DataFrames and Series for tabular and single-column data
- File operations (CSV, Excel, SQL)
- Missing data handling
- Grouping, filtering, pivoting, merging
- Time-series data support

Example:
```
import pandas as pd

df = pd.read_csv('employees.csv')
df['total'] = df['salary'] + df['bonus']
grouped = df.groupby('department')['total'].mean()
```

14. How do you integrate Python with big data tools like Apache Spark?
Use **PySpark**, the Python API for Apache Spark.
Example:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Example").getOrCreate()
df = spark.read.csv('data.csv', header=True, inferSchema=True)

df.select("name", "age").show()
```
Use it for distributed data processing (ETL, ML) across clusters.

Data Modeling and Architecture

• Difference between a database, data warehouse, and data lake

- **Database**: Designed for OLTP (Online Transaction Processing) — used for real-time operations like CRUD (Create, Read, Update, Delete). Think of apps like banking or e-commerce where you store structured data (like customer info, transactions).
- **Data Warehouse**: Optimized for OLAP (Online Analytical Processing) — stores structured, cleaned, and transformed data for reporting and analytics. Think of Power BI dashboards sourcing data from a warehouse.
- **Data Lake**: A massive storage repo that can hold raw data (structured, semi-structured, or unstructured). Ideal for big data processing and advanced analytics or ML use cases.

• What are fact and dimension tables in data modeling?

- **Fact Table**: Stores quantitative data (like sales amount, profit) — it's where metrics live.
- **Dimension Table**: Stores descriptive attributes (like product name, date, store, etc.) — used for slicing and dicing data in reports. Think of a sales report: sales amount = fact; product, date, and customer = dimensions.

• Difference between ER modeling and dimensional modeling

- **ER Modeling (Entity-Relationship)**: Used in transactional systems; focuses on reducing redundancy using normalization.
- **Dimensional Modeling**: Used in analytical systems; designed for fast query performance. It uses facts and dimensions and includes star/snowflake schemas.

• What is the process of normalization, and why is it required?

Normalization is breaking down large tables into smaller ones to eliminate data redundancy and improve data integrity. It's required in OLTP systems to avoid anomalies (like update/delete anomalies) and to ensure data is stored efficiently.

• Explain the deployment architecture of a data pipeline involving ADF, Databricks, and ADLS

A common architecture looks like this:

1. **ADF** extracts data from source (on-prem or cloud) and loads it into **ADLS (raw zone)**.
2. **Databricks** processes data (transformations, joins, aggregations) and writes clean and enriched data back to **ADLS (cleansed and curated zones)**.
3. ADF may load curated data into **Synapse** or a **data warehouse** for reporting.
4. **Monitoring and logging** are handled in ADF via pipeline monitoring and alerts.

• Build a simple data warehouse for a grocery store

You'd design:

- **Fact_Sales**: transaction_id, product_id, store_id, date_id, quantity_sold, revenue
- **Dim_Product**: product_id, name, category, brand

- **Dim_Store**: store_id, location, store_type
- **Dim_Date**: date_id, date, week, month, year This would be a classic **star schema**.

● What are the different types of data schemas, and how do you choose the right one for your data model?

- **Star Schema**: Simplified, easy to understand, good for reporting (fact in the middle, surrounded by dimensions).
- **Snowflake Schema**: Normalized dimension tables; useful when dimensions have hierarchies.
- **Galaxy/Fact Constellation**: Multiple fact tables sharing dimension tables. Choose based on:
- Query performance
- Complexity of dimensions
- Scalability and maintenance needs

● How do you handle slowly changing dimensions (SCD) in data modeling?

- **SCD Type 1**: Overwrite old data. Simple but you lose history.
- **SCD Type 2**: Keep history by adding a new row with versioning and effective dates.
- **SCD Type 3**: Add new columns to hold previous values (rarely used). Usually, Type 2 is preferred in analytics for historical analysis.

● Explain the concept of denormalization and when it should be used

Denormalization combines tables to reduce joins and improve read performance. It's used in OLAP systems where query speed is more critical than storage efficiency. It’s common in data warehouses or reporting databases.

● How do you design a star schema for a sales reporting system?

Start with the **Fact_Sales** table that includes metrics. Link it to dimension tables like:

- **Dim_Customer**
- **Dim_Product**
- **Dim_Date**
- **Dim_Store** The star schema will enable fast aggregation (e.g., sales by product, region, or time).

● What is data mart, and how does it differ from a data warehouse?

A **Data Mart** is a subset of a data warehouse tailored for a specific business unit (e.g., sales, marketing). It’s more focused and faster to build, while a **Data Warehouse** is enterprise-wide and more comprehensive.

● How do you ensure data consistency and integrity in a distributed data architecture?

- Use **primary/foreign key constraints** during ingestion
- Employ **validation rules** during transformation (e.g., NULL checks, data types)
- Use **schema validation**, **data quality checks**, and **ETL error handling**
- Implement **data lineage** and **audit logging**

● Describe the role of metadata in data modeling and data architecture

Metadata is “data about data.” It describes structure (columns, types), lineage (where data came from), and business context. Good metadata management enables data governance, discovery, impact analysis, and auditing.

● How do you optimize data models for performance and scalability?

- Use **appropriate indexing** and **partitioning**
- Choose **star schema** over snowflake when performance is critical
- Avoid unnecessary joins
- Archive old data periodically

- Use **columnar storage formats** (like Parquet) for analytics workloads

• Explain the use of surrogate keys in dimensional modeling
Surrogate keys are unique, system-generated identifiers (e.g., auto-increment integers) used in dimension tables instead of business keys. They help manage SCDs easily and avoid issues with changing natural keys.

• How do you implement a data governance framework in a data lake environment?
- Set **naming conventions** and **folder structure**
- Use **Unity Catalog**, **Azure Purview**, or similar tools for **data cataloging**
- Define **roles and permissions** using RBAC or ACLs
- Implement **data quality rules**, **data classification**, and **data lineage**
- Monitor usage and enable **auditing and compliance logging**

Other Questions

1. What is DAG in Spark, and how does it work?
In Apache Spark, a **DAG (Directed Acyclic Graph)** represents the sequence of computations performed on data. When you execute a series of transformations on an RDD (Resilient Distributed Dataset), Spark constructs a DAG to track these operations. The DAG is divided into stages based on transformations:✍
- **Stages**: Each stage consists of tasks that can be executed in parallel. There are two types:✍
  - **Shuffle Stages**: Involve data shuffling between nodes.✍
  - **Non-Shuffle Stages**: Do not involve data movement between nodes.✍

The DAG scheduler in Spark divides the graph into these stages and assigns tasks to worker nodes for execution. ✂cite turn0search0☎✍

2. Explain Spark architecture (Driver vs Worker).
Apache Spark follows a master-slave architecture comprising:✍
- **Driver**: The central coordinator that converts user programs into tasks and schedules them across the cluster. It runs the main program and tracks the execution.✍
- **Workers (Executors)**: Processes running on worker nodes that execute tasks assigned by the driver. Each executor runs in its own Java process and handles task execution and data storage.✍

The driver communicates with multiple executors to distribute and parallelize tasks across the cluster. ✂cite turn0search1☎✍

3. What is the difference between wide transformations and narrow transformations in Spark?
In Spark, transformations are categorized based on data dependencies:✍
- **Narrow Transformations**: Each partition of the parent RDD is used by at most one partition of the child RDD. Examples include map() and filter(). These transformations do not require data shuffling and are more efficient.✍
- **Wide Transformations**: Multiple child partitions may depend on one or more partitions of the parent RDD, necessitating data shuffling across the cluster. Examples include groupByKey() and reduceByKey(). These operations are more resource-intensive due to the data movement involved.✍

Understanding these helps in optimizing Spark jobs by minimizing expensive wide transformations when possible. ✂cite turn0search2☎✍

4. How does fault tolerance work in ADF and Databricks?
Both Azure Data Factory (ADF) and Azure Databricks incorporate fault tolerance mechanisms:✍

- **Azure Data Factory**: ADF ensures fault tolerance by allowing activities to retry upon failure. You can configure the retry policy with parameters like count and interval. Additionally, ADF can handle transient failures by implementing robust error-handling logic within pipelines.✍
- **Azure Databricks**: Databricks enhances fault tolerance through features like job clusters from pools, which provide workload isolation and faster cluster creation. This setup supports auto-termination upon job completion and offers resilience against failures.
  ✂cite turn0search3☎✍

5. What challenges have you faced in managing large datasets (e.g., 3TB+ files)?
Managing large datasets presents several challenges:✍

- **Storage Requirements**: Large volumes necessitate substantial storage capacity, leading to increased costs and infrastructure complexity.✍
- **Performance Issues**: As data volume grows, retrieval and processing times can increase, resulting in slower and less responsive analytics.✍
- **Data Quality**: Ensuring consistency and accuracy becomes more complex with larger datasets, requiring robust validation mechanisms.✍

Addressing these challenges involves implementing efficient storage solutions, optimizing data processing workflows, and establishing stringent data quality protocols. ✂cite turn0search4☎✍

6. How do you implement CI/CD pipelines for deploying ADF and Databricks solutions?
Implementing CI/CD pipelines involves automating the deployment and integration processes:✍

- **Azure Data Factory**: Utilize Azure DevOps to create build and release pipelines that automate the deployment of ADF pipelines and related resources.✍
- **Azure Databricks**: Implement CI/CD by integrating Databricks notebooks with version control systems like Git. Use Azure DevOps pipelines to automate testing and deployment of notebooks and other Databricks artifacts. ✂cite turn0search5☎✍

This approach ensures consistent and reliable deployments across environments.✍

7. How to improve the performance of a Spark job?
Enhancing Spark job performance can be achieved through several strategies:✍

- **Use DataFrames/Datasets**: Prefer DataFrames or Datasets over RDDs as they offer built-in optimizations.✍
- **Optimize Transformations**: Use coalesce() instead of repartition() to reduce the number of partitions without full data shuffle.✍
- **Avoid UDFs**: User Defined Functions can hinder performance; leverage Spark's built-in functions when possible.✍
- **Caching**: Cache intermediate results when they are reused multiple times to avoid recomputation.✍
- **Reduce Shuffling**: Minimize operations that trigger shuffling, such as groupByKey(), by using alternatives like reduceByKey(). ✂cite turn0search6☎✍

Implementing these practices can lead to significant performance gains.✍

8. What is the use of Delta Lake, and how does it support ACID transactions?
Delta Lake is an open-source storage layer that brings ACID (Atomicity, Consistency, Isolation, Durability) transactions to data lakes:✍

- **ACID Transactions**: Delta Lake ensures that all operations on data are atomic, consistent, isolated, and durable. This means that each transaction is completed fully or not at all, data remains consistent before and after transactions, transactions are isolated from each other, and once a transaction is committed, it remains

✔ **What is a Unity Catalog in Databricks?**

Unity Catalog is a unified governance solution for all data and AI assets in the Databricks Lakehouse platform. It centralizes and standardizes access control, auditing, and lineage tracking across workspaces.

Key Features:

- **Centralized metadata management:** One catalog for multiple workspaces.
- **Fine-grained access control:** Manage access down to the column and row level using ANSI SQL.
- **Data lineage:** Automatically track how data moves and transforms from raw to refined.
- **Support for multiple storage accounts:** Unlike legacy Hive metastore, it supports federated access.

Use case: If you're working in a multi-team or enterprise setup, Unity Catalog helps you enforce consistent data governance and security across projects.

✔ **Explain the role of Integration Runtimes in hybrid scenarios (on-prem to cloud).**

Integration Runtime (IR) is the compute infrastructure used by **Azure Data Factory (ADF)** to perform data movement, transformation, and dispatch activities.

There are **three types**:

1. **Azure IR** – Used for cloud-to-cloud integration.
2. **Self-hosted IR** – Installed on-premises to support **on-prem to cloud** data movement.
3. **Azure-SSIS IR** – Runs SSIS packages in ADF.

In hybrid scenarios (e.g., moving data from on-prem SQL Server to ADLS):

- You install **Self-hosted IR** within your on-prem network.
- It securely connects to your local data sources and moves data to the cloud without exposing them publicly.

Why important: Enables secure and scalable movement of data across hybrid environments, ensuring compliance and performance.

✔ **How to design an end-to-end data pipeline architecture?**

Designing an **end-to-end data pipeline** involves stitching together multiple components to move, transform, store, and analyze data.

Typical Flow:

1. **Ingestion Layer (Raw Layer):**
    - Tools: ADF, Azure Event Hubs, Kafka.
    - Ingest data from databases, APIs, logs, IoT devices, etc.
2. **Storage Layer:**
    - Use **Azure Data Lake Storage Gen2** or Blob Storage for raw data.
    - Store it in organized folders (like Bronze layer in medallion architecture).
3. **Processing/Transformation Layer:**
    - Use **Databricks or Synapse Spark** for batch/stream processing.
    - Apply cleansing, deduplication, joins, aggregations.
4. **Serving Layer (Gold/Curated):**
    - Store processed data in Synapse SQL, Power BI datasets, or Delta tables.
    - Ready for reporting, dashboards, and downstream consumption.
5. **Orchestration and Scheduling:**
    - Use **ADF pipelines** or **Azure Synapse Pipelines** to schedule and monitor jobs.
6. **Monitoring & Logging:**
    - Leverage **Azure Monitor**, **Log Analytics**, or built-in pipeline monitoring tools.

Security & Governance: Use Key Vault for secrets, Unity Catalog for data access control, and Purview for metadata management.

✔ **Snowflake-Specific Features**

❓ How does it compare to Delta Lake or other data lakes?

Feature	Snowflake	Delta Lake (Databricks)
Storage Architecture	Decoupled compute & storage (multi-cluster shared data)	Built on top of Parquet files with transactional support
ACID Transactions	Fully supported	Supported via Delta Lake
Performance	Auto-scaling compute, fast query performance	High performance with tuning options
Ease of Use	SQL-first, no infrastructure management	More flexible but requires Spark knowledge
Data Sharing	Seamless cross-account sharing (Snowflake Data Sharing)	Data sharing possible but more manual
Streaming Support	Limited native support	Excellent streaming with Structured Streaming

❓ When would you choose Snowflake over other platforms?

- You want a **fully-managed, SQL-friendly** cloud data warehouse.
- Your team has limited **Spark expertise**.
- You need **multi-region or cross-cloud deployment** (Snowflake runs on AWS, Azure, and GCP).
- For use cases needing **instant compute scaling**, **automatic tuning**, and **zero maintenance**.

✔ Difference between Azure Logic Apps, Azure Functions, and Azure Data Factory

Feature	Azure Logic Apps	Azure Functions	Azure Data Factory
Use Case	Workflow automation (no/low code)	Serverless functions/code execution	Data movement and transformation
Code vs No-Code	Low-code visual designer	Code-first (C#, Python, JS)	Low-code UI with some scripting
Triggers	Event-based, HTTP, schedules	HTTP trigger, timer, queue, blob	Time-based triggers, event triggers
Best for	Integrations between systems (e.g., email, Teams, SharePoint)	Light compute logic, custom scripts	ETL/ELT data workflows
Integration	300+ connectors	Extend workflows with custom logic	Works well with data sources (SQL, ADLS, Blob, etc.)

✔ Azure Synapse Analytics: How does it compare to Databricks?

Feature	Azure Synapse	Databricks
Core Purpose	Analytics and reporting (SQL + Spark)	Unified data processing & ML
Languages Supported	T-SQL, Spark, .NET	Python, Scala, SQL, R
Best For	SQL-heavy data warehousing	Big data + AI/ML + streaming
Compute Models	Dedicated SQL pools, Serverless SQL, Spark pools	Fully managed Spark clusters
Ease of Use	Familiar to SQL users, tight Power BI integration	More suited for data scientists/engineers
Cost Efficiency	Cheaper for small workloads (serverless SQL)	More flexible scaling for heavy Spark jobs
Real-time	Basic support	Excellent (via Spark Structured

Feature	Azure Synapse	Databricks
Streaming		Streaming)
Delta Lake Support	Basic via Spark pools	Native, robust Delta Lake support

Summary:

- Choose **Synapse** if your team is SQL-savvy and needs a BI-centric platform.
- Choose **Databricks** if you're doing **advanced data engineering, ML**, or real-time big data processing.

✔ Azure DevOps & CI/CD

1. How to use Azure DevOps for CI/CD pipelines?
Azure DevOps provides tools for managing code (Repos), building pipelines (Pipelines), and releasing applications (Releases). You can:

- Push your code to Azure Repos or GitHub.
- Set up a **CI pipeline** to automatically build, test, and validate code when changes are committed.
- Set up a **CD pipeline** to deploy code to environments like Azure Data Factory, Databricks, Synapse, or Azure Functions.
- Use YAML files or classic UI-based editors to define pipeline steps.

2. What are the key features of Azure DevOps?

- **Azure Repos** – Git version control.
- **Azure Pipelines** – CI/CD automation.
- **Azure Boards** – Agile project tracking (Kanban, Scrum).
- **Azure Artifacts** – Package management (e.g., NuGet, npm).
- **Test Plans** – Manual and automated testing support.

3. How do you implement CI/CD in Azure DevOps?

- CI: Automatically trigger builds when code changes are committed.
- CD: Automatically deploy the build artifacts to target environments (QA, Prod).
- Use environments, approvals, and gates to control deployments.

4. **Explain the role of pipelines in Azure DevOps.**

   Pipelines automate your software delivery process. They:

- Build code.
- Run unit and integration tests.
- Deploy applications or services.
- Can be written in YAML or configured visually (classic pipelines).

5. **How do you manage dependencies and versioning in Azure DevOps?**

- Use **Azure Artifacts** for managing and hosting packages.
- Use semantic versioning (e.g., 1.0.0) in your builds.
- Use requirements.txt (Python) or package.json (Node.js) to lock dependencies.

⚙ Azure Functions
6. What are Azure Functions, and how do they differ from traditional web services? Azure Functions are serverless – you don’t manage infrastructure. They're:

- Event-driven: Triggered by HTTP requests, blobs, queues, timers, etc.
- Short-lived, stateless, and scalable.

- Compared to traditional web APIs, Functions are lighter, cost-effective, and easier to deploy for specific tasks.

7. How do you deploy and manage Azure Functions?
- Deploy using Azure DevOps, GitHub Actions, or manually via VS Code.
- Monitor and manage using Azure Portal (logs, performance metrics, triggers).
- Use Application Insights for debugging and monitoring.

8. Use cases for Azure Functions in data engineering:
- Triggering a Databricks job on blob upload.
- Cleaning or validating incoming data before storage.
- Sending notifications or alerts.
- Ingesting and transforming data before writing to storage.

🔐 Azure Key Vault

9. How do you secure secrets and keys using Azure Key Vault?
- Store secrets (passwords, connection strings), certificates, and encryption keys securely.
- Integrate with Azure services to avoid hardcoding secrets in pipelines or notebooks.
- Access controlled via Azure RBAC and Access Policies.

10. How do you integrate Azure Key Vault with other services?
- Azure Data Factory: Linked Service → "Use Azure Key Vault" to pull secrets.
- Azure Functions: Use Managed Identity to access secrets securely.
- Azure Databricks: Mount secrets via Databricks-backed secret scope or Azure Key Vault-backed scope.

📊 Azure Synapse Analytics

11. What are the benefits of Azure Synapse for big data processing?
- Unified platform: Combines data warehousing and big data.
- Supports T-SQL and Spark workloads.
- Serverless and dedicated compute options.
- Integration with Power BI, ADF, and ADLS for end-to-end pipelines.

12. How does Synapse integrate with other Azure services?
- **ADF** for orchestration.
- **ADLS Gen2** as data lake.
- **Power BI** for visualization.
- **Azure Purview** for data governance.
- **Azure Monitor** for observability.

13. Difference between dedicated SQL pool and serverless SQL pool:
- **Dedicated SQL Pool**: Pre-allocated compute for high-performance queries, paid per hour.
- **Serverless SQL Pool**: Query data on-demand (pay-per-query), great for ad hoc analysis or querying raw files (e.g., CSV, Parquet).

14. How do you monitor and optimize performance in Synapse?
- Use **SQL Activity Monitoring** and **Query Insights**.
- Monitor Spark jobs via Spark UI.
- Use **Dynamic Management Views (DMVs)** to check memory, I/O, CPU.
- Partition and distribute large tables to reduce shuffles.

15. Describe the process of data ingestion in Synapse.

- Ingest data via Synapse Pipelines (like ADF).
- Pull from sources like Azure SQL DB, Blob Storage, API, etc.
- Store in data lake (ADLS) or directly load into SQL pools.

16. Best practices for data partitioning and distribution:

- Choose distribution method: Hash, Round Robin, or Replicated.
- Partition large fact tables on frequently filtered columns.
- Avoid data skew in hash distributions.

17. How to implement security and compliance in Synapse?

- Use **RBAC**, **network isolation**, **firewalls**, and **Private Endpoints**.
- Enable **auditing and threat detection**.
- Encrypt data at rest and in transit.
- Integrate with **Azure Purview** for data cataloging.

🔄 Azure Data Factory (ADF)

18. Concept of data integration and transformation in ADF:

- ADF lets you move data between sources and transform it using:
  - Mapping Data Flows (GUI-based ETL)
  - Databricks Notebooks
  - Stored Procedures
  - Custom activities (e.g., Python scripts)

19. Error handling and retry mechanisms in ADF:

- Use the **retry policy** in activities.
- Catch errors with **Try-Catch** pattern using **If Condition** and **Execute Pipeline** activities.
- Log errors in SQL table or send alerts via email/webhook.

20. Triggers and schedules in ADF:

- **Schedule Trigger**: Run pipelines at specific times (e.g., every hour).
- **Tumbling Window Trigger**: Time-sliced execution with dependency chaining.
- **Event-Based Trigger**: Run pipelines on blob creation or deletion.

📈 Azure Monitor

21. How do you use Azure Monitor to track performance in Azure services?

- Collect telemetry from services like ADF, Synapse, Functions, etc.
- Use **Log Analytics** workspace to write Kusto queries for analysis.
- Set up alerts based on metrics (e.g., pipeline failures, high latency).
- Integrate with **Application Insights** for detailed diagnostics.

Gopi Rayavarapu!

www.linkedin.com/in/gopi-rayavarapu-5b560020a