

INFOSYS

Data Engineer

Interview Questions

ctc: 13-17 LPA

yoe: 3+

1) What is the difference between a job cluster and an interactive cluster in databricks ?

Databricks: Job cluster vs Interactive cluster

Purpose & workflow

- **Interactive cluster:** Long-lived cluster you (and teammates) attach notebooks to for exploration, development, ad-hoc analysis, and debugging.
- **Job cluster:** Ephemeral cluster created **only** for a job/task run and terminated when the run finishes. Ideal for production/CI workloads.

Lifecycle & cost

- **Interactive:** Stays up until you terminate it (or it hits auto-termination). You pay while it's running—even when idle (until auto-terminate kicks in).
- **Job:** Spins up for the run and shuts down automatically afterward → you only pay for the run window.

Configuration & governance

- **Interactive:**

- Created in the Clusters UI (or via API) and reused.
 - You attach/upgrade libraries manually; good for iterative dev.
 - Often governed by **Cluster Policies** that restrict node types, DBR versions, libraries, etc.
- **Job:**
 - Cluster definition is embedded in the Job (UI or JSON/API). Each run gets a clean environment.
 - Libraries specified in the job/task (e.g., Maven/PyPI/JAR) or via **init scripts**.
 - Easier to enforce reproducibility and avoid “it works on my cluster” drift.

State & caching

- **Interactive:** Caches (e.g., Delta cache, executor JVM state) can persist across notebook commands—faster iteration.
- **Job:** Fresh state each run; caches vanish after termination—more predictable.

Concurrency & sharing

- **Interactive:** Multiple users can attach notebooks; good for pair debugging and workshops.
- **Job:** Tied to a job run; no ad-hoc notebook sharing during execution.

Reliability

- **Interactive:** More manual control; easier to inspect live executors/drivers for debugging.
- **Job:** Better for **idempotent** production pipelines; clean cluster means fewer “dirty state” surprises.

Best practice

- Develop on **Interactive**, promote to **Job** for scheduled/production runs.
 - Use **Cluster Pools** to shorten spin-up time for job clusters.
 - Lock down with **Cluster Policies**, pin **DBR** versions, and treat job clusters as **immutable** infra.
-

2) How to copy all tables from one source to the target using metadata - driven pipelines in ADF ?

ADF: Copy all tables from a source to a target with a metadata-driven pipeline

Below is a proven design that scales, is easy to govern, and supports both **full** and **incremental** loads.

A) Metadata model (in Azure SQL / Synapse SQL / dedicated config DB)

Create a **control table** that lists what to copy and how:

```
CREATE TABLE etl.TableCopyConfig (
    ConfigId      INT IDENTITY PRIMARY KEY,
    SourceLinkedService NVARCHAR(100),
    SourceDb        NVARCHAR(128),
    SourceSchema    NVARCHAR(128),
    SourceTable     NVARCHAR(128),
    SinkLinkedService NVARCHAR(100),
    SinkDb          NVARCHAR(128),
    SinkSchema      NVARCHAR(128),
    SinkTable       NVARCHAR(128),
    LoadType        VARCHAR(20) CHECK (LoadType IN ('Full','Incremental')),
    WatermarkColumn NVARCHAR(128) NULL, -- e.g. ModifiedDate, LastUpdatedTs
    LastWatermarkValue NVARCHAR(128) NULL, -- store last/max value loaded
    KeyColumns      NVARCHAR(4000) NULL, -- comma-separated keys for upsert
    PreCopyScript    NVARCHAR(MAX) NULL, -- optional DDL/DML
    PostCopyScript   NVARCHAR(MAX) NULL, -- optional DDL/DML
    IsActive        BIT NOT NULL DEFAULT 1,
    Notes           NVARCHAR(4000) NULL
);
```

Optional: add AdditionalProperties (JSON) for column mappings, file formats, partition hints, etc.

B) Parameterized datasets

1. **Source dataset (Azure SQL example)**
 - Parameters: pSchema, pTable, pDb, pLS
 - Table: @{{format('{0}.{1}', pipeline().parameters.pSchema, pipeline().parameters.pTable)}}
 - Linked service name is dynamic via a **global parameter** or **switch** (or use separate datasets per LS if you prefer simplicity).
2. **Sink dataset (Azure SQL / Synapse / ADLS)**
 - Parameters mirror source (pSchema, pTable, pDb, pLS) or for lake targets: pContainer, pFolder, pFileName.

C) Pipeline outline

Parameters (at pipeline level): none required, but you can add run-level filters (e.g., pOnlySchema, pOnlyTable).

Activities

1. **Lookup** (name: GetConfig):
Query active rows:
2. SELECT * FROM etl.TableCopyConfig WHERE IsActive = 1

Set **First row only = false**.

3. **ForEach** over @activity('GetConfig').output.value
Set batchCount (e.g., 5–10) to control parallelism.

Inside ForEach:

- **If** item().LoadType == 'Full'
 - Optional **Stored Procedure / Script** activity (name: PreFull) to run item().PreCopyScript or a generated TRUNCATE TABLE [schema].[table] on the sink.
 - **Copy Data** activity (name: CopyFull)

- **Source:** table or SELECT * FROM [SourceSchema].[SourceTable]
 - **Sink:** sink table/file; enable **Auto mapping** when schemas are identical.
 - Tuning: parallelCopies, sqlWriterUseTableLock = true, bulk insert/batching settings.
- **Else (Incremental)**
 - **Set Variable** (or directly in source query) to build predicate:
 - @concat
 - 'SELECT * FROM [', item().SourceSchema, '].[', item().SourceTable, '] ',
 - 'WHERE ', item().WatermarkColumn, ' > ',
 - case(isNumeric(item().LastWatermarkValue),
item().LastWatermarkValue, concat("", item().LastWatermarkValue,
""))
 -)
 - **Copy Data** activity (name: CopyIncr) with above **Source query**.
 - **Stored Procedure** activity (name: UpdateWatermark) to update etl.TableCopyConfig.LastWatermarkValue with:
 - For datetime watermark:
 - UPDATE c
 - SET LastWatermarkValue = @MaxWatermark
 - FROM etl.TableCopyConfig c
 - WHERE c.ConfigId = @ConfigId;

where @MaxWatermark is computed in-run (e.g., use **Query** on source to get MAX(WatermarkColumn) before/after copy; or use **Copy activity**'s output.rowsRead and a separate Lookup for MAX()).

- **Optional PostCopy:** run item().PostCopyScript (index/create stats).

Notes & gotchas

- **Schema drift:** If the sink is SQL, consider a nightly DDL sync (compare sys.columns) or allow the copy to target a lake in **delta/Parquet** to absorb drift.
- **Upserts:** For incremental **MERGE**, swap the sink to a **Stored Procedure** that merges from a **staging table** you load in CopyIncr.
- **Performance:**
 - Use **Self-Hosted IR** when the source is on-prem/private.
 - For Synapse targets, prefer staging & **PolyBase/COPY INTO**.
 - Tune batchCount, parallelCopies, and partition the source query (e.g., MOD on key) if it's huge.
- **Observability:** Write run logs to a table etl.RunLog capturing PipelineRunId, ConfigId, rows, duration, errors.
- **Security:** Parameterize linked services, use **Managed Identity** with AAD auth to source/sink, never embed credentials in metadata.

Minimal ADF dynamic content examples

- Source table:
`@{format('{0}.{1}', item().SourceSchema, item().SourceTable)}`
- Sink table:
`@{format('{0}.{1}', item().SinkSchema, item().SinkTable)}`
- Incremental predicate:
`@{format('{0} > {1}', item().WatermarkColumn,
if>equals(item().IsWatermarkNumeric, true), item().LastWatermarkValue, concat("",
item().LastWatermarkValue, "")))}`

3) How do you implement data encryption in Azure SQL Database ?

Azure SQL Database: Implementing data encryption

Think in **layers**: *at rest, in transit, and in use/column-level*.

A) At rest: Transparent Data Encryption (TDE)

- **What:** Encrypts data files and backups on disk. No app changes. TempDB is covered.
- **Default:** New Azure SQL DBs typically have TDE **ON** with **Microsoft-managed keys**.
- **Customer-Managed Keys (CMK)** with **Azure Key Vault (BYOK)**:
 1. Create/choose an RSA key in **Azure Key Vault**.
 2. Enable **Managed Identity** on the Azure SQL **server** (logical server).
 3. Grant that identity **Key Vault** permissions (RBAC or access policy: get, wrapKey, unwrapKey, list).
 4. Set the server's **TDE Protector** to the Key Vault key (Portal, PowerShell, or CLI).
 5. (Optional) Enable **auto-rotation** in Key Vault; SQL picks up new versions.
- **PowerShell (illustrative)**
 - # Set CMK as TDE protector
 - Set-AzSqlServerTransparentDataEncryptionProtector `
 - -ResourceGroupName "rg" `
 - -ServerName "sql-svr" `
 - -Type AzureKeyVault `
 - -KeyId "https://<kv>.vault.azure.net/keys/<keyname>/<keyver>"
 -
 - # Ensure database encryption is ON
 - Set-AzSqlDatabaseTransparentDataEncryption `
 - -ResourceGroupName "rg" -ServerName "sql-svr" -DatabaseName "appdb" -State "Enabled"
- **T-SQL (Microsoft-managed keys)**
 - -- On Azure SQL DB, often already ON; but you can enforce:
 - ALTER DATABASE [appdb] SET ENCRYPTION ON;

B) In transit: TLS encryption

- **Server setting:** Enforce **Minimum TLS Version** (e.g., 1.2) on the SQL server.
- **Client/connection string** (ADO.NET example):
 - Encrypt=True; TrustServerCertificate=False; MultipleActiveResultSets=False;
- **Firewall & Private access:** Prefer **Private Link** + AAD auth to reduce exposure.

C) Column-level / In-use protection

Option 1: Always Encrypted (client-side encryption)

- **What:** Sensitive columns are encrypted **in the client driver**; the database/engine only sees ciphertext.
- **Keys:**
 - **Column Master Key (CMK):** stored in Key Vault or local cert store.
 - **Column Encryption Key (CEK):** stored in the database, wrapped by the CMK.
- **Encryption types:**
 - **Deterministic:** allows equality joins/group by; risk of pattern leakage.
 - **Randomized:** strongest; no equality operations.
- **Workflow:**
 1. Provision CMK in Key Vault; grant app/service principal access.
 2. Use SSMS/SQLPackage/.NET to create CMK/CEK metadata and encrypt target columns.
 3. Ensure app uses AE-capable drivers (ODBC 17+, JDBC 8+, .NET 4.6+), with Column Encryption Setting=Enabled.
- **T-SQL metadata (illustrative):**
 - CREATE COLUMN MASTER KEY [CMK_AKV]
 - WITH (KEY_STORE_PROVIDER_NAME = 'AZURE_KEY_VAULT',
 - KEY_PATH = 'https://<kv>.vault.azure.net/keys/<keyname>/<version>');
 -
 - CREATE COLUMN ENCRYPTION KEY [CEK1]
 - WITH VALUES (COLUMN_MASTER_KEY = [CMK_AKV],

- ALGORITHM = 'RSA_OAEP',
- ENCRYPTED_VALUE = 0x...); -- produced by client tool
-
- -- Example encrypted column
- CREATE TABLE dbo.Customer (
 - CustomerId INT PRIMARY KEY,
 - SSN NVARCHAR(11) COLLATE Latin1_General_BIN2
 - ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = [CEK1], ENCRYPTION_TYPE = DETERMINISTIC, ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'),
 - Name NVARCHAR(100) NOT NULL
 -);
- **Secure Enclaves (AEv2)**: Enables in-place encrypt/decrypt and richer ops (range comparisons) while keeping data protected in an enclave. Supported in Azure SQL Database with compatible SKUs/drivers.

Option 2: Cell-level encryption (T-SQL functions)

- CREATE SYMMETRIC KEY + EncryptByKey / DecryptByKey. More manual key management; generally superseded by Always Encrypted for app data.

Not encryption but related:

- **Dynamic Data Masking**: Obfuscation for query results; doesn't protect data at rest/in transit.
- **Row-Level Security**: Access control, not encryption.

D) Operational tips

- Store secrets in **Managed Identity + Key Vault**; avoid connection-string passwords.
- Periodically **rotate keys** (CMK/Key Vault versions); validate app behavior.
- Monitor with **Azure Defender for SQL** / Vulnerability Assessments.
- Test backup/restore with CMK-protected TDE to ensure you can access the protector key.

- For ETL tools (ADF, Databricks) interacting with **Always Encrypted** columns, verify driver support; if not feasible, stage encrypted data end-to-end or use enclave-enabled flows.
-

4) Write a Python code to generate fibonacci series ?

Python code to generate Fibonacci series

Iterative approach

```
def fibonacci_series(n):  
    """  
    Generate Fibonacci series up to n terms.  
    """  
  
    a, b = 0, 1  
  
    series = []  
  
    for _ in range(n):  
        series.append(a)  
        a, b = b, a + b  
  
    return series  
  
# Example: first 10 terms  
print(fibonacci_series(10))
```

Output

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Recursive approach (not optimal for large n)

```
def fib(n):  
    if n <= 1:  
        return n
```

```
return fib(n-1) + fib(n-2)
```

```
print([fib(i) for i in range(10)])
```

Generator (memory-efficient)

```
def fibonacci_gen(n):
```

```
    a, b = 0, 1
```

```
    for _ in range(n):
```

```
        yield a
```

```
        a, b = b, a + b
```

```
print(list(fibonacci_gen(10)))
```

👉 In interviews, it's good to mention **iterative is most efficient** ($O(n)$, $O(1)$ space).

5) What are the best practises for managing and optimizing storage costs in ADLS ?

Best practices for managing and optimizing storage costs in ADLS (Azure Data Lake Storage)

Storage costs in **ADLS Gen2** depend on **capacity, transactions, redundancy, and lifecycle policies**.

Here are key practices:

A) Data Lifecycle & Tiering

- Use **Lifecycle Management policies** to automatically move data:
 - **Hot tier** → frequently accessed data.
 - **Cool tier** → infrequently accessed (30+ days).
 - **Archive tier** → rarely accessed (180+ days).
- Example: Move logs older than 30 days to Cool, and older than 180 days to Archive.

B) Partitioning & Structuring

- Organize data into **hierarchical folder structures** (e.g., /year=2025/month=09/day=04/) to optimize queries and reduce scanning.
- Avoid too many **small files** (increase transaction costs + query inefficiency). Use compaction strategies (e.g., merge Parquet files in Databricks).

C) Redundancy Strategy

- Choose redundancy based on criticality:
 - **LRS** (Locally Redundant Storage) = cheapest, single region.
 - **ZRS** (Zone Redundant Storage) = balanced availability.
 - **GRS/RA-GRS** = geo-redundancy; higher cost.
- Don't overpay for redundancy where not needed (e.g., dev/test data).

D) Monitoring & Alerts

- Use **Azure Cost Management + Monitor** to:
 - Track capacity growth & transaction patterns.
 - Set alerts for sudden cost spikes (caused by misconfigured pipelines).

E) Access Patterns & Optimization

- Cache frequently accessed reference data in compute (Databricks cache, Synapse materialized views).
- Minimize unnecessary read/write operations (especially overwriting large files).
- For ETL: write **append-only Parquet/Delta** instead of frequent full overwrites.

F) Governance & Cleanup

- Apply **Retention Policies** → auto-delete obsolete datasets.
- Regularly audit unused datasets and purge stale data.

👉 Example interview phrase:

"We optimize ADLS costs using lifecycle tiering, minimizing small files, choosing appropriate redundancy, and monitoring with cost alerts while enforcing retention policies."

6) How do you implement security measures for data in transit and at rest in azure ?

Implementing security measures for data in transit and at rest in Azure

Data protection in Azure = **Defense in Depth**. Cover both **at rest** and **in transit**.

A) Data at Rest

1. Encryption

- By default, all Azure storage (ADLS, Blob, SQL, CosmosDB) is encrypted with **Storage Service Encryption (SSE)** using **Microsoft-managed keys**.
- For extra control, use **Customer-Managed Keys (CMK)** stored in **Azure Key Vault**.
- Optionally enable **Double Encryption** for high-security workloads.

2. Transparent Data Encryption (TDE)

- For **Azure SQL DB/Synapse**, encrypts database, log, and backups automatically.
- Supports BYOK (Bring Your Own Key) with Key Vault.

3. Always Encrypted (column-level)

- Protects sensitive fields (SSN, credit card) from even DBAs.

4. Disk-level Encryption

- Azure Disk Encryption for VMs (BitLocker for Windows, DM-Crypt for Linux).
-

B) Data in Transit

1. TLS/SSL Encryption

- All Azure services enforce **HTTPS** (TLS 1.2+).
- Block unencrypted connections (Secure transfer required = Enabled in storage accounts).

2. Private Network Access

- Use **Private Endpoints** (Private Link) to connect services over Azure backbone, avoiding public internet.
- Restrict access via **NSGs, Firewall rules, and Service Endpoints**.

3. VPN / ExpressRoute

- For hybrid scenarios, secure connections from on-prem via VPN/ExpressRoute with encryption.

4. Client & App Security

- Use **Managed Identity** instead of embedding credentials.
 - Rotate secrets with **Key Vault + RBAC**.
-

C) Best Practices

- Apply **Role-Based Access Control (RBAC)** and **Least Privilege**.
 - Enable **Azure Defender for Cloud** for advanced threat detection.
 - Enable **Diagnostic Logs** and audit all access to data.
 - Regularly rotate keys/certificates and monitor with **Key Vault Key Rotation Policies**.
-

7) Describe the role of triggers and schedules in ADF?

Role of Triggers and Schedules in ADF

In Azure Data Factory (ADF), **pipelines** don't run by themselves—you need **triggers** to initiate execution.

◆ Types of Triggers

1. Schedule Trigger

- Runs pipelines at a **predefined time/frequency**.
- Example: Run pipeline daily at **2 AM** for data refresh.
- Supports recurrence (minutes, hours, days, weeks, months).

2. Tumbling Window Trigger

- Time-bound, non-overlapping, and **guarantees exactly once** execution for each time window.
- Useful for **time-sliced data ingestion** (e.g., hourly logs).
- Supports **retry policy** and **backfilling** (process past windows if missed).

3. Event-based Trigger

- Fires pipeline when an **event occurs** in **Blob Storage / ADLS** (e.g., file creation or deletion).
- Example: Trigger pipeline when a new CSV lands in a container.

4. Manual (On-demand)

- No trigger, user runs pipeline manually from UI/API.
-

◆ Best Practices

- Use **Schedule** for batch jobs, **Tumbling Window** for streaming-style data slices, and **Event** for real-time ingestion.
 - Combine with **parameters** to make pipelines reusable across triggers.
 - Always add **concurrency and retry settings** to avoid duplicate runs.
-

8) Write a sql query to find employees with no manager assigned ?

SQL query to find employees with no manager assigned

Let's assume we have an **Employees** table:

Employees(

 EmpID INT,

 EmpName VARCHAR(100),

 ManagerID INT

)

Solution 1: IS NULL

```
SELECT EmpID, EmpName  
FROM Employees  
WHERE ManagerID IS NULL;
```

👉 Finds employees where no manager is assigned.

Solution 2: Using LEFT JOIN

If manager details exist in the same table:

```
SELECT e.EmpID, e.EmpName  
FROM Employees e  
LEFT JOIN Employees m  
ON e.ManagerID = m.EmpID  
WHERE e.ManagerID IS NULL OR m.EmpID IS NULL;
```

- Captures employees with no manager or where the manager record is missing.

Interview tip:

Mention both cases—NULL manager (top-level employees like CEO) **and** manager ID not present in the table (data quality issue).

9) How do you implement data deduplication in pyspark ?

Implementing Data Deduplication in PySpark

Data duplication is common in big data. PySpark provides multiple approaches.

Let's say we have a DataFrame df:

```
from pyspark.sql import SparkSession  
  
spark = SparkSession.builder.appName("dedup").getOrCreate()  
  
data = [  
    (1, "Alice", "HR"),  
    (2, "Bob", "IT"),  
    (3, "Charlie", "HR"),  
    (4, "David", "IT"),  
    (5, "Eve", "HR"),  
    (6, "Frank", "IT"),  
    (7, "Grace", "HR"),  
    (8, "Hank", "IT"),  
    (9, "Ivy", "HR"),  
    (10, "Jack", "IT")]
```

```
(2, "Bob", "IT"),
```

```
(3, "Alice", "HR"),
```

```
(4, "David", "Finance"),
```

```
(5, "Bob", "IT")
```

```
]
```

```
df = spark.createDataFrame(data, ["EmpID", "Name", "Dept"])
```

```
df.show()
```

◆ **1. Remove exact duplicates**

```
df_distinct = df.distinct()
```

```
df_distinct.show()
```

- Removes rows where all column values are identical.
-

◆ **2. Remove duplicates based on specific columns**

```
df_no_dupes = df.dropDuplicates(["Name", "Dept"])
```

```
df_no_dupes.show()
```

- Keeps only the first occurrence of each (Name, Dept) pair.
-

◆ **3. Deduplication using Window function (keep latest by timestamp/ID)**

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

```
import pyspark.sql.functions as F
```

```
# Suppose we want the latest record per employee based on EmpID
```

```
windowSpec = Window.partitionBy("Name", "Dept").orderBy(F.desc("EmpID"))
```

```
df_dedup = (df.withColumn("row_num", F.row_number().over(windowSpec))
    .filter(F.col("row_num") == 1)
    .drop("row_num"))
```

```
df_dedup.show()
```

- Ensures deterministic deduplication by keeping the latest row in each group.
-

 **Interview-ready summary:**

"In PySpark, we handle duplicates using distinct() for full row uniqueness, dropDuplicates() when specific columns define uniqueness, and window functions when we need to deduplicate based on business rules like latest timestamp or highest ID."

10) Find 2nd Highest Salary in SQL

Let's assume a table:

Employees(

 EmpID INT,

 EmpName VARCHAR(100),

 Salary INT

)

 **Method 1: LIMIT / OFFSET (MySQL, Postgres, SQL Server 2012+ with OFFSET)**

```
SELECT DISTINCT Salary
```

```
FROM Employees
```

```
ORDER BY Salary DESC
```

```
LIMIT 1 OFFSET 1;
```

 **Method 2: TOP + Subquery (SQL Server)**

```
SELECT MAX(Salary) AS SecondHighestSalary
```

```
FROM Employees  
WHERE Salary < (SELECT MAX(Salary) FROM Employees);
```

 **Method 3: Window Function (ANSI SQL, recommended)**

```
SELECT Salary  
FROM (  
    SELECT Salary,  
        DENSE_RANK() OVER (ORDER BY Salary DESC) AS rnk  
    FROM Employees  
) t  
WHERE rnk = 2;
```

- **DENSE_RANK()** ensures if the highest salary repeats, the next distinct value is picked as 2nd highest.
- **ROW_NUMBER()** would skip duplicates, **RANK()** would skip gaps. Best to mention these differences.

 **Interview tip:** Always mention **multiple approaches**.

11) How do you implement incremental load in databricks ?

Implementing Incremental Load in Databricks

Incremental load = only load **new or changed records**, instead of full refresh.
This saves cost, improves performance, and avoids overwriting large datasets.

 **Common Approaches in Databricks**

A) Using Watermark Column (e.g., last_updated_timestamp)

1. Identify a column (e.g., last_updated, modified_on) that tracks changes.
2. Store the **max value loaded** from last run in a checkpoint table or metadata file.

3. Next run → read only rows WHERE last_updated > last_checkpoint.

```
# Assume we store checkpoint in Delta table

checkpoint=
spark.read.table("etl_metadata").filter("table_name='orders'").select("last_watermark").fir
st()[0]

# Read incremental data from source

incremental_df = (spark.read.format("jdbc")
    .option("url", "jdbc:sqlserver://...")
    .option("dbtable", f"(SELECT * FROM orders WHERE last_updated > '{checkpoint}'")
t")
    .load())

# Write into Delta

incremental_df.write.format("delta").mode("append").save("/mnt/delta/orders")

# Update checkpoint

new_checkpoint = incremental_df.agg({"last_updated": "max"}).collect()[0][0]

spark.sql(f"UPDATE etl_metadata SET last_watermark='{new_checkpoint}' WHERE
table_name='orders'"")
```

B) Using Delta Lake MERGE (Upserts)

If records can be **updated as well as inserted**, use **MERGE INTO** with a unique key.

```
from delta.tables import DeltaTable
from pyspark.sql.functions import col
```

```
# Load target Delta table
```

```
target = DeltaTable.forPath(spark, "/mnt/delta/customers")

# Load source incremental data

source = spark.read.format("parquet").load("/mnt/raw/customers/2025-09-04")

# Merge (upsert)

(target.alias("t")
 .merge(source.alias("s"), "t.customer_id = s.customer_id")
 .whenMatchedUpdateAll()
 .whenNotMatchedInsertAll()
 .execute())
```

C) Using Change Data Capture (CDC)

- For **SQL Server / Cosmos DB**, Databricks can read **CDC-enabled tables** or event streams (Kafka/Event Hub).
- Use **Auto Loader** with **cloudFiles** option to process only new files.

```
df = (spark.readStream
      .format("cloudFiles")
      .option("cloudFiles.format", "json")
      .load("/mnt/raw/orders"))
```

Interview-ready summary:

"In Databricks, incremental load is implemented by tracking watermarks (last updated column), using Delta Lake MERGE for upserts, or leveraging CDC/Auto Loader for streaming sources. We store the checkpoint in Delta/metadata tables and only process data newer than the last watermark."

12) Describe the role of Azure Key Vault in securing sensitive data?

Role of Azure Key Vault in Securing Sensitive Data

Azure Key Vault (AKV) is a centralized secret management service that protects **keys, secrets, and certificates**.

Key Capabilities

1. Secrets Management

- Stores connection strings, passwords, API tokens, certificates.
- Applications fetch secrets securely instead of hardcoding.

2. Key Management

- Stores cryptographic keys (RSA, ECC, HSM-backed).
- Used for **data encryption** (e.g., TDE/Always Encrypted keys in SQL, storage encryption keys).

3. Certificates Management

- Central place for SSL/TLS certs with auto-rotation.

Integration with Azure Services

- **Azure SQL / Synapse / CosmosDB:** Use **Customer-Managed Keys (CMK)** from Key Vault.
- **Storage Accounts / ADLS:** Encryption-at-rest with CMK.
- **Databricks / ADF:** Store and retrieve DB credentials securely.
- **App Services / Functions:** Access secrets via **Managed Identity** → no credentials in code.

Security Features

- **Access Control:** Uses **Azure RBAC** & Key Vault Access Policies.
- **Managed Identities:** Apps authenticate without secrets.

- **Auditing:** Logs every access to Azure Monitor/Log Analytics.
 - **High availability:** Geo-redundant, backed by HSMs (FIPS 140-2 Level 2+).
-

 **Interview-ready summary:**

"Azure Key Vault secures sensitive data by centrally managing keys, secrets, and certificates. It integrates with Azure SQL, ADLS, and ADF for encryption-at-rest and with apps via Managed Identity for secret retrieval. It ensures least-privilege access, audit logging, and automated rotation of secrets and certificates."

13) Explain the concept of shuffling in Spark

Shuffling is the process of redistributing data across partitions and nodes in a Spark cluster. It usually happens when Spark needs to group, join, or aggregate data by a key.

◆ **When does shuffling occur?**

- `groupByKey()`, `reduceByKey()`, `aggregateByKey()`, `join()`, `distinct()`, `repartition()`.
- Anytime data from one partition needs to be moved to another partition.

◆ **How it works:**

1. Spark writes intermediate data from map tasks to disk.
2. Data is transferred (over the network) to reducers based on partitioning logic (e.g., hash partitioning).
3. Reducers read shuffled data and perform aggregation/joins.

◆ **Impacts of shuffling:**

- **Performance overhead** – involves disk I/O, network transfer, and serialization.
- **Data skew issues** – if keys are unevenly distributed, some partitions get more data, leading to long job execution times.

◆ **Best Practices to minimize shuffling:**

- Use map-side operations like `mapPartitions()` or `reduceByKey()` instead of `groupByKey()`.
- Repartition data smartly (e.g., `repartition()` vs `coalesce()`).

- Use partitioning strategies (broadcast join when one dataset is small).

Example (shuffling in join):

```
df1 = spark.createDataFrame([(1, "A"), (2, "B")], ["id", "val1"])
```

```
df2 = spark.createDataFrame([(1, "X"), (3, "Y")], ["id", "val2"])
```

```
# This join causes shuffle because Spark must move rows with same key together
```

```
result = df1.join(df2, "id", "inner")
```

14) What are the key considerations for designing scalable pipelines in ADF?

When building scalable pipelines in **Azure Data Factory (ADF)**, you need to consider **performance, cost, maintainability, and reliability**.

◆ **Key Considerations:**

1. Pipeline Design & Modularity

- Break pipelines into reusable, modular components (metadata-driven design).
- Use **parameterized pipelines** instead of hardcoding.

2. Data Movement Efficiency

- Use **copy activity with staging** for large data (Blob Storage → Synapse).
- Optimize **integration runtime** (self-hosted vs Azure).

3. Scalability

- Enable **parallelism** (use ForEach with batchCount).
- Partition large datasets and process in parallel.

4. Incremental Loads

- Use **watermark columns** or **Change Data Capture (CDC)** for incremental data movement.
- Avoid full reloads when possible.

5. Performance Optimization

- Enable **PolyBase** or **COPY INTO** for large data loads into Synapse.
- Use **compression** (snappy, gzip) to reduce I/O.

6. Monitoring & Logging

- Use **Activity Run Output** for logging metadata (rows copied, time taken).
- Enable **Azure Monitor / Log Analytics** for alerting.

7. Security & Compliance

- Store secrets in **Azure Key Vault**, not inside ADF.
- Use **Managed Identity** for authentication.

8. Cost Optimization

- Minimize unnecessary data movement (process data in place when possible).
- Schedule pipelines efficiently (avoid running too frequently without need).

Example: Metadata-driven incremental pipeline

- A control table in SQL stores source/target mappings.
- ADF pipeline reads this table and dynamically loads tables in parallel.
- Uses watermark column (last_updated) for incremental loading.