

1 What do you understand by Change Data Capture (CDC)?

✓ Answer:

Change Data Capture (CDC) is a technique used to identify and capture only the changes (INSERT, UPDATE, DELETE) in source data, rather than reloading the entire dataset.

It ensures incremental data loading into data warehouses or lakes, improving performance and reducing costs.

📌 Use Cases:

- Real-time ETL pipelines
- Replicating databases to a data warehouse
- Streaming ingestion with tools like Debezium or Azure Data Factory

2 Can you explain more about Databricks and its architecture?

✓ Answer:

Databricks is a unified analytics platform built on top of Apache Spark. It provides collaborative notebooks, automated cluster management, and seamless integration with cloud services like Azure, AWS, and GCP.

🔧 Architecture Overview:

- **Driver Node:** Manages SparkContext and job scheduling
- **Worker Nodes (Executors):** Perform distributed computation
- **Cluster Manager:** Manages cluster lifecycle (e.g., Azure Databricks uses YARN or Databricks native)
- **Notebooks/UI:** Used for interactive development, visualization, and job orchestration

📌 Databricks is ideal for big data processing, machine learning workflows, and data engineering pipelines.

3 Why shouldn't we use inferred schema for large datasets?

✓ Answer:

When using an inferred schema, Databricks (or Spark) reads multiple rows to guess the column data types.

⚠️ Issues with Inferred Schema:

- Performance overhead: Scans large files to infer types
- Inaccuracy: May assign wrong data types if sampling misses edge cases

- Scalability: Slower pipeline initialization on large datasets

✅ **Best Practice:**

Use explicit schema definitions for large or critical datasets to ensure speed and consistency.

4 **What is a prolonged secrets scope in Databricks?**

✅ **Answer:**

A prolonged secrets scope in Databricks refers to secrets (credentials, tokens, keys) stored for long-term or repeated access.

🔑 **Databricks supports:**

- Databricks-backed scopes (managed by Databricks)
- Azure Key Vault-backed scopes (external integration for security)

📌 **Use Cases:**

- Mounting ADLS securely
 - Accessing APIs or databases
 - Reusability across pipelines without exposing credentials in code
-

5 **What are the parameters used while mounting storage in Databricks?**

✅ **Answer:**

To mount Azure Data Lake or Blob Storage in Databricks, use the following:

python

CopyEdit

```
dbutils.fs.mount(  
    source = "wasbs://<container>@<storage-account>.blob.core.windows.net/",  
    mount_point = "/mnt/<mount-name>",  
    extra_configs = {  
        "fs.azure.account.key.<storage-account>.blob.core.windows.net": "<access-key>"  
    }  
)
```

📌 **Parameters:**

- **source:** Storage container URL
- **mount_point:** Path in DBFS
- **extra_configs:** Authentication config (account key, SAS token, or OAuth)

 Always store keys in secret scopes, not in plaintext.


6 Do stored procedures exist in Databricks?

Answer:

No, traditional stored procedures (like in SQL Server) are not natively supported in Databricks SQL.

However, you can:

- Use notebooks or Delta Live Tables for procedural logic
- Leverage user-defined functions (UDFs) and SQL widgets to build parameterized logic
- Create SQL dashboards with filters

 For procedural operations: Write logic in PySpark, Scala, or SQL inside notebooks/pipelines.


7 What are different types of window functions?

Answer:

Window functions allow you to perform row-wise computations across partitions.

Common Types:

- ROW_NUMBER() – Assigns a unique row number
- RANK() / DENSE_RANK() – Ranking with or without gaps
- LAG() / LEAD() – Fetch previous/next row values
- NTILE(n) – Distributes rows into n buckets
- SUM() OVER(PARTITION BY...) – Rolling aggregations

 Used in scenarios like top-N, sessionization, cumulative sums, and time-based analytics.

8 What are different types of JOINS in Databricks?

Answer:

Databricks supports standard SQL-style and optimized joins:

◆ Join Types:

- INNER JOIN – Matching rows only
- LEFT JOIN – All from left, matched from right
- RIGHT JOIN – All from right, matched from left
- FULL OUTER JOIN – All rows from both sides
- LEFT ANTI JOIN – Only rows from left not in right
- LEFT SEMI JOIN – Only rows from left that exist in right

✦ Use broadcast joins for small dimension tables to optimize shuffles.

9 What are UDFs in PySpark? How do they impact performance and use cases?

✓ Answer:

A UDF (User Defined Function) is a custom function written in Python (or Scala/Java) to apply non-native transformations on Spark columns.

python

CopyEdit

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

```
@udf
```

```
def upper_case(s): return s.upper()
```

● Performance Warning:

- Slower than native Spark SQL functions
- Break Catalyst optimizer pipeline
- Require serialization (Python to JVM)

✓ Use Spark SQL built-in functions (like `when`, `regexp_replace`) whenever possible.

10 Explain Databricks Delta Live Tables (DLT), and when would you use batch vs. streaming?

✓ Answer:

Delta Live Tables (DLT) is a managed ETL framework in Databricks for building reliable, declarative pipelines using SQL or Python.

◆ **Key Benefits:**

- Built-in monitoring & data quality checks
- Easy integration with Delta Lake
- Handles schema enforcement, change propagation, and orchestration

📦 **Batch Mode:**

- For daily/hourly pipelines
- Use when low-latency isn't needed

📡 **Streaming Mode:**

- For real-time or near real-time processing
- Use with tools like Kafka or Event Hubs

📌 DLT simplifies pipeline creation and maintenance using "LIVE" tables.