**Azure Data Bricks Technical Questions   
  
1. What is Azure Databricks, and how does it integrate with other Azure services like Azure Data Lake, Azure Synapse Analytics, and Azure Machine Learning?**

Azure Databricks is a unified analytics platform based on Apache Spark, designed for big data processing and machine learning. It integrates with Azure Data Lake for scalable storage, Azure Synapse for analytics, and Azure ML for model training and deployment. Native connectors enable seamless, secure data movement across services.

**2. Explain the role and differences between an All-Purpose cluster and a Job cluster in Azure Databricks. When would you choose one over the other?**

All-Purpose clusters are interactive and used for collaborative development, allowing multiple users and notebooks. Job clusters are ephemeral and created specifically for automated job runs, terminating after execution. Use All-Purpose for exploration and Job clusters for scheduled ETL or production pipelines.

**3. How does Spark SQL work in Azure Databricks, and what is the Catalyst optimizer? Can you describe its role in query optimization?**

Spark SQL allows querying structured data using SQL within Databricks. The Catalyst optimizer is the query optimizer in Spark that transforms SQL queries into optimized execution plans through rule-based and cost-based strategies, improving performance without user intervention.

**4. In Azure Databricks, how do you perform data reading and writing operations using Spark DataFrames? Provide examples of how to read from CSV, Parquet, and Delta files.**

df\_csv = spark.read.csv("/mnt/data/file.csv", header=True, inferSchema=True)

df\_parquet = spark.read.parquet("/mnt/data/file.parquet")

df\_delta = spark.read.format("delta").load("/mnt/data/delta\_table")

df\_csv.write.format("delta").save("/mnt/output/")

**5. What are the key differences between DataFrames and RDDs in Spark? How does Spark's DataFrame API improve performance over RDDs?**

RDDs offer fine-grained control and are resilient but lack optimizations. DataFrames provide a higher-level API with schema, enabling Spark SQL and Catalyst optimizations. DataFrames improve performance through logical and physical query plans, reducing execution time and memory usage

* **abstraction:**
  + **RDD (Resilient Distributed Dataset):** Low-level API with no schema; you work with objects directly.
  + **DataFrame:** High-level API with schema; similar to a table in a relational database.
* **Ease of Use:**
  + **RDDs:** Require more code for transformations and actions.
  + **DataFrames:** Offer concise syntax using SQL-like operations.
* **Optimization:**
  + **RDDs:** No automatic optimizations; you manage execution manually.
  + **DataFrames:** Optimized by **Catalyst Optimizer** and **Tungsten engine** for better execution planning and memory management.
* **Performance:**
  + **DataFrames** outperform RDDs due to optimizations like predicate pushdown, bytecode generation, and efficient memory usage.

✅ **In short:** Use **DataFrames** for better performance and simpler code; use **RDDs** when you need fine-grained control or custom processing logic.

**6. How do you register a DataFrame as a temporary view in Spark SQL? Can you execute SQL queries on that view?**

df.createOrReplaceTempView("sales\_view")

spark.sql("SELECT \* FROM sales\_view WHERE revenue > 10000")

**7. Explain the concept of caching and persistence in Spark. How do these techniques optimize the performance of iterative algorithms or repeated queries?**

Caching and persistence store intermediate DataFrames/RDDs in memory (or disk) to avoid recomputation. This boosts performance for iterative jobs or reused datasets. Use .cache() for memory storage and .persist(StorageLevel) for fine-grained control.

**8. What are broadcast joins in Spark, and how do they improve the performance of joins when dealing with large datasets? Can you explain a scenario where you would use them?**

Broadcast joins replicate a small table to all worker nodes, avoiding expensive shuffles. Use them when joining a large fact table with a small dimension table. For example, joining a 100GB transactions table with a 1MB country mapping table.

**9. In a machine learning workflow in Azure Databricks, what libraries and tools would you typically use for data preparation, model training, and evaluation? How does Databricks support distributed training for ML models?**

**10. When dealing with a performance bottleneck in a Spark job running on Azure Databricks, what diagnostic steps would you take to identify the issue, and how would you optimize the performance?**

**mount point** is a **shortcut or symbolic link** that allows a Databricks workspace to access and interact with external storage systems like **Azure Data Lake Storage (ADLS)** or **Amazon S3** as if they were part of the local file system (/mntdirectory).

**What is Delta Table versioning?"** in an interview setting:

**Interviewer:** *Can you explain how Delta Table versioning works in Databricks?*

**Candidate (You):**

Yes, absolutely.

Delta Lake supports **versioning** by maintaining a **transaction log** in the \_delta\_log directory of the Delta table. Every time a write operation—like an INSERT, UPDATE, DELETE, or MERGE—is performed, Delta creates a new **version** of the table.

This enables powerful features like:

* **Time travel**: You can query the table as it was at a specific point in time or a specific version using the VERSION AS OF or TIMESTAMP AS OF clauses.

For example:

sql

CopyEdit

SELECT \* FROM delta.`/mnt/datalake/sales` VERSION AS OF 5;

* **Audit and debugging**: Using the DESCRIBE HISTORY command, I can see who made changes, when, and what operations were performed. This is extremely useful for data governance.

sql

CopyEdit

DESCRIBE HISTORY delta.`/mnt/datalake/sales`;

* **Rollback**: I can easily restore the table to a previous version using the RESTORE command if needed.

Overall, Delta versioning brings **ACID compliance** and **traceability** to data lakes, making it easier to manage large-scale data processing workflows.

### **OneLake**

* **What it is**: The **unified, logical data lake** storage layer for all data in Microsoft Fabric.
* **Think of it as**: A single, organization-wide **data lake** that connects all workloads—Lakehouse, Warehouse, Power BI, Real-Time Analytics, etc.
* **Key Feature**: Every Fabric workspace is a folder in OneLake, and data is stored once but made accessible in multiple ways (via **Shortcuts**).

### ✅ **2. Data Lake**

* **What it is**: A centralized **storage repository** that holds raw, structured, semi-structured, and unstructured data (e.g., JSON, CSV, images).
* **Usage**: Ideal for **big data** analytics, machine learning, and data science.
* **In Fabric**: OneLake serves as the Data Lake using **Delta Lake format**.

### **3. Data Warehouse**

* **What it is**: A **relational database** optimized for **structured data**, historical analysis, and complex SQL queries.
* **Usage**: BI reporting, dashboards, decision support systems.
* **In Fabric**: Synapse Data Warehouse supports T-SQL, built for scalability and performance with ACID compliance.

### **5. Delta Lake**

* **What it is**: An **open-source storage layer** that brings **ACID transactions**, **versioning**, and **schema enforcement**to data lakes.
* **In Fabric**: All Lakehouse and Warehouse tables use **Delta format** underneath.
* **Why it matters**: Enables **reliable**, performant **analytics and incremental processing**.

### **. Medallion Architecture**

* **Definition**: A layered data architecture pattern typically used in Lakehouses.
* **Layers**:
  + **Bronze**: Raw, ingested data (minimal processing)
  + **Silver**: Cleaned, deduplicated, joined data
  + **Gold**: Business-ready, aggregated, analytical data
* **Usage in Fabric**: Implemented via **Notebooks**, **Dataflows**, and **Pipelines** in Lakehouse and Warehouse.

### **7. Lakehouse**

* **What it is**: A **hybrid architecture** that combines the scalability of a data lake with the performance and schema of a data warehouse.
* **In Fabric**: Lakehouses support Delta tables and can be queried using SQL or Spark, accessible directly by Power BI.
* **Key Benefit**: Unified processing and storage with **no data duplication**.

### **ommon Transformations in Databricks (Spark)**

| **Transformation Type** | **Example Functions** | **Description** |
| --- | --- | --- |
| **Narrow** | map(), filter(), flatMap() | Operate on a single partition – no shuffling. |
| **Wide** | groupBy(), join(), reduceByKey(), distinct() | Involve shuffling data across partitions. |
| **Set Operations** | union(), intersect(), subtract() | Perform set logic on two datasets. |
| **Aggregation** | groupBy().agg(), groupByKey(), reduceByKey() | Used to summarize data. |
| **Sorting** | sortBy(), orderBy() | Reorders the data. |
| **Partitioning** | repartition(), coalesce() | Adjusts number of partitions. |
| **Schema Ops** | select(), withColumn(), drop() | Modify or select columns. |
| **Window Functions** | row\_number(), rank(), lead(), lag() | Used for advanced analytics over windows. |