

Here are **detailed answers** to your **Databricks | PySpark | SQL technical interview questions** – each with full context to help you prepare effectively for Data Engineer roles:

Technical Round – Databricks | PySpark | SQL

1. What is database normalization, and why is it important?

Answer:

Normalization is the process of organizing data in a relational database to eliminate redundancy and improve data integrity. It involves dividing a database into two or more tables and defining relationships between them.

It ensures:

- Efficient data storage
- Reduced data anomalies (insert/update/delete)
- Easier maintenance and updates

Example: Breaking down a single flat Employee table with repeated department names into separate Employee and Department tables.

2. How would you explain Databricks and its architecture to a beginner?

Answer:

Databricks is a unified data analytics platform built on Apache Spark, designed for big data and machine learning workloads.

Architecture:

- **Control Plane:** Managed by Databricks – handles job scheduling, workspace UI, notebooks, etc.
 - **Compute Plane:** Where clusters are created in your cloud (Azure/AWS/GCP), and actual data processing occurs.
Databricks separates compute and storage, enabling scalability and performance.
-

3. How does PySpark integrate within the Databricks platform?

Answer:

PySpark is the Python API for Apache Spark, and it is fully supported in Databricks notebooks.

Users can:

- Write Spark transformations/actions in Python
 - Use Spark DataFrames and Spark SQL in Python
 - Leverage Databricks' interactive notebooks to visualize data
Databricks also handles session management, cluster provisioning, and Python package integration.
-

4. What types of clusters are available in Databricks?

 **Answer:**

- **Interactive Cluster (All-purpose):** Used for development, ad-hoc analysis, notebooks.
 - **Job Cluster:** Automatically created and terminated for running production jobs/pipelines.
 - **High Concurrency Cluster:** Optimized for multiple users running queries simultaneously (used in BI use cases).
You can configure autoscaling, worker nodes, and use Photon engine for performance.
-

5. What are the different schema types, and why is inferred schema risky for large datasets?

 **Answer:**

- **Inferred Schema:** Automatically determines column data types based on a sample of the data.
- **Explicit/Defined Schema:** Manually provides the structure during data loading.

Inferred schema is risky for large datasets because:

- It samples a limited portion (default 1000 rows), which can miss column type inconsistencies
 - Leads to incorrect type inference (e.g., treating numeric columns as string)
 - Slows performance during read operation
-

6. What is a secret scope, and how do you use it securely in Databricks?

 **Answer:**

A **secret scope** in Databricks is a secure way to store credentials like access keys, database

passwords, and tokens.

Secure usage:

- Created using Databricks CLI or UI
 - Accessed using `dbutils.secrets.get(scope="name", key="key")`
 - Secrets are **never hardcoded** in notebooks or shared with unauthorized users
They integrate with Azure Key Vault for enterprise-grade security.
-

🔴 7. How do you mount storage in a workspace? What are the key parameters involved?

✅ **Answer:**

Mounting allows Databricks users to interact with cloud storage (e.g., Azure Data Lake) as if it's part of the Databricks File System (DBFS).

Key function:

python

CopyEdit

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

Key parameters:

- **source:** storage URL
 - **mount_point:** path in DBFS
 - **extra_configs:** auth credentials (use secrets here for security)
-

🔴 8. Can you use stored procedures in Databricks SQL?

✅ **Answer:**

Databricks supports **stored procedures** via **Delta Live Tables (DLT)** or **SQL procedural language** with **CREATE PROCEDURE** syntax in newer versions.

However, traditional RDBMS-style stored procedures are **limited in functionality**.

Databricks encourages using **notebooks**, **workflows**, and **dbt-like modular SQL scripts** for logic reuse.

📌 9. What's your approach to loading structured data (e.g., name, address, contact) into DataFrames?

✅ Answer:

You can use:

python

CopyEdit

```
df = spark.read.format("csv").option("header", True).schema(schema).load("/path/file.csv")
```

or for structured input like JSON/Parquet:

python

CopyEdit

```
df = spark.read.json("/path/file.json")
```

- Define schema explicitly for consistency
- Clean and cast data types as needed
- Validate nulls, duplicates, and formats

📌 10. Share the SQL query to fetch records where today's temperature is higher than the previous day

✅ Answer:

sql

CopyEdit

```
SELECT day, temperature
```

```
FROM (
```

```
    SELECT day, temperature,
```

```
           LAG(temperature) OVER (ORDER BY day) AS prev_temp
```

```
    FROM temperature_data
```

```
) temp_diff
```

```
WHERE temperature > prev_temp;
```

Explanation: Uses LAG() window function to compare today's temperature with the previous day.

📌 11. What are the types of window functions available in PySpark?

✅ Answer:

- **Ranking functions:** row_number(), rank(), dense_rank()
- **Analytic functions:** lag(), lead(), first(), last()
- **Aggregate over window:** sum(), avg(), min(), max() over a sliding window
Example:

python

CopyEdit

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

```
from pyspark.sql.functions import row_number
```

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

```
df.withColumn("row_num", row_number().over(windowSpec)).show()
```

📌 12. How to fetch the 5th highest salary using SQL?

✅ Answer:

Method 1 – Using Subquery with DISTINCT and LIMIT:

sql

CopyEdit

```
SELECT DISTINCT salary
```

```
FROM employee
```

```
ORDER BY salary DESC
```

```
LIMIT 1 OFFSET 4;
```

Method 2 – Using DENSE_RANK() Window Function:

sql

CopyEdit

```
SELECT * FROM (
```

```
    SELECT *, DENSE_RANK() OVER (ORDER BY salary DESC) AS rnk
```

FROM employee

) ranked

WHERE rnk = 5;

Note: Choose based on whether duplicates in salary are acceptable or not.