**Altimetrik Data Engineer Interview Guide – Experienced 3+**

**Round 1: Core Concepts and Project Experience**

**1. Self-Introduction and Project Explanation**

The interview began with a standard introduction where the interviewer asked:

 **Tell me about yourself and your experience.**

**Explain your recent projects in detail. Tips**:

 Highlight your most relevant projects, focusing on your role, tools/technologies used, challenges faced, and solutions implemented.

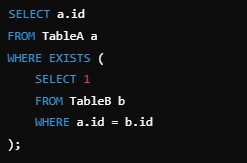
 Clearly differentiate between AWS and GCP projects, focusing on unique aspects.

**2. Conceptual Questions**

1. **Semi-Join**

**Definition**: A semi-join retrieves rows from a table where matching rows exist in another table but does not return the matching rows.

**Example**:



2. **Cross-Join**

**Definition**: A cross-join produces a Cartesian product of two tables.

**Tip**: Mention its use case in scenarios where all combinations of rows are needed.

3. **Primary Key vs. Secondary Key**

**Primary Key**: Uniquely identifies a row in a table and cannot be null.

**Secondary Key**: A key used to improve query performance, often as an index.

4. **Foreign Key**

**Definition**: A field in one table that links to the primary key in another table, establishing relationships.

5. **Minimum Age Query**

**SQL**:



**3. Airflow Questions**

1. **Initiating a DAG**

Create a Python file in the DAGs folder and define a DAG using airflow.models.DAG.

2. **Operators in Airflow**

Operators are tasks in a DAG, such as PythonOperator, BashOperator, DummyOperator, etc.

3. **Concept of Task**

A task is a unit of work in a DAG, defining what operation needs to be performed.

**Round 2: Advanced Concepts and Scenario-Based Questions**

**1. Big Data Tools**

1. **Hadoop Commands for Get and Merge** hadoop fs -get /source/path /local/path hadoop fs -merge /source/path /target/file

2. **Hadoop Architecture**

Discuss NameNode, DataNode, and Secondary NameNode, emphasizing

HDFS and YARN

3. **Spark Context vs. Spark Session**

**Spark Context**: Entry point for older Spark versions, managing the Spark application.

**Spark Session**: Unified entry point introduced in Spark 2.0, encapsulating both Spark Context and SQL Context.

**2. Specific Scenarios and Concepts**

1. **Null Value Handling in a Single Column**

Use fillna or replace in PySpark:



2. **YARN**

A resource manager for Hadoop, allocating resources and scheduling tasks across the cluster.

3. **Map vs. FlatMap**

**Map**: Applies a function to each element, producing one output per input.

**FlatMap**: Applies a function to each element and flattens the results.

4. **Sqoop Incremental Import**

Use --incremental append or --incremental lastmodified for incremental imports.

5. **Left Anti Join**

Use case: Finding rows in the left table that don’t have matching rows in the right table.

**Example**:



**3. Cloud and Spark-Related Questions**

1. **Web API Reading**

Use libraries like requests or urllib in Python for API data ingestion, then transform and load it into the target system.

2. **Scala Traits**

**Definition**: Traits are similar to interfaces in Java, allowing multiple inheritance.

3. **Executor Memory in Spark**

Stores RDD partitions, caches data, and performs computations.

4. **Broadcasting in Spark**

Used to efficiently distribute large read-only data across all nodes.

**4. Databricks and Delta Lake**

1. **dbutils Function**

Used for managing files, secrets, and jobs in Databricks.

**Example**: dbutils.fs.mv(source, destination)

2. **Moving Files in DBFS**

Command: dbutils.fs.mv('/source/path', '/destination/path')

3. **Job Cluster in Databricks**

Create a cluster via the Databricks UI or CLI and specify the cluster mode as

"Job".

4. **Lazy Evaluation in Spark**

Spark evaluates transformations only when an action (like count or collect) is triggered.

5. **Managed vs. External Tables**

**Managed Table**: Spark manages the metadata and data storage.

**External Table**: Data is stored outside Spark, and only metadata is managed.

6. **Delta Lakehouse Architecture**

Combines the benefits of data lakes and data warehouses. Supports ACID

transactions, schema enforcement, and real-time analytics.

7. **Bronze/Silver/Gold Layers**

**Bronze**: Raw data.

**Silver**: Cleaned and validated data.

**Gold**: Aggregated and business-ready data.

8. **Deployment Process**

Use CI/CD pipelines to move code from Dev to QA/Prod.

9. **Scheduling Jobs in Databricks**

Use the Databricks Jobs UI to define tasks, set dependencies, and schedule triggers.