**Building an ETL Testing Framework for PySpark SQL Queries**

**Overview**

This document outlines the process of designing an ETL testing framework for validating SQL queries within PySpark-based ETL scripts. The objective is to dynamically test SQL queries stored in Python variables, ensuring correctness using sample data before executing them in a production environment.

**Steps to Implement the Framework**

1. **Define the Testing Approach**
   * The goal is to validate SQL queries defined in PySpark scripts.
   * Queries should be extracted from Python variables rather than executing the full main() function.
   * Sample data is used to validate query execution and ensure correctness.
2. **Modify the PySpark Configuration Script**
   * Convert inline SQL queries within main() to module-level variables.
   * Ensure SQL query variables are accessible outside the function scope.
   * Avoid executing queries directly in main() to enable independent testing.
3. **Setting Up the Testing Framework**
   * Develop a test class for automated validation of SQL execution.
   * Load sample CSV data into Spark temporary views.
   * Dynamically import SQL queries from the PySpark script.
   * Execute queries and validate output.
4. **Handling Sample Data for Testing**
   * Store sample data in CSV files formatted with section headers.
   * Each file should contain multiple table definitions using the format [table\_name].
   * The test framework reads these files and loads data into Spark temporary views.
5. **SQL Query Execution and Validation**
   * Retrieve SQL queries from the PySpark script.
   * Validate query syntax before execution.
   * Execute queries and create temporary views.
   * Verify output data consistency and correctness.
6. **Logging and Debugging**
   * Implement logging for test execution status.
   * Log errors and unexpected behaviors to facilitate debugging.
   * Ensure meaningful error messages for missing data or failed queries.
7. **Running the Tests**
   * Use a test runner script to execute unit tests.
   * Initialize a Spark session within the test script.
   * Validate output tables against expected row counts.
   * Ensure no critical errors occur during SQL execution.

**Conclusion**

This framework allows systematic validation of PySpark SQL queries before production deployment. By structuring SQL execution separately from main(), testability is improved, reducing the risk of production failures. The approach ensures modularity, scalability, and maintainability of ETL testing in PySpark-based pipelines.