# Databricks Homework - 02 - File Formats and Data Storage

SciEncephalon AI Associates Program

Databricks

## Introduction

This assignment builds on your understanding of **big data storage formats and schema management in Apache Spark**. You will work with different file formats, explore schema evolution strategies, leverage **Delta Lake for transactional storage**, and optimize data storage using **partitioning and bucketing.**

You will be using **open datasets** in **Databricks** and working with **PySpark** to analyze file formats and apply schema management techniques.

## Prerequisites

- **Python (PySpark) and SQL knowledge**.
- **Familiarity with Spark DataFrames and RDDs** (from Homework 1).
- **A working Databricks environment** with a cluster supporting Delta Lake.

## Assignment Instructions

- **Submit a well-documented Jupyter Notebook (.ipynb) or Databricks Notebook (.dbc)** with your code and explanations.
- **Email your submission early** to allow time for feedback.
- **Be prepared to share your screen** in the next session and explain your work.
- You may **use online resources**, but **ensure you understand and can explain your approach**.

## Exercises

### Exercise 1: File Formats in Spark

**Objective**: Compare different file formats (CSV, JSON, Parquet, ORC, Delta, Avro) for structured data storage.

**Tasks:**

1. **Load a Sample Dataset:**
    - Use a sample dataset such as `/databricks-datasets/airlines/part-00000` or load a CSV file of your choice.
    - Read it into a Spark DataFrame using `spark.read.csv()`
2. **Save the Data in Different Formats:**
    - Save the DataFrame in **CSV, JSON, Parquet, ORC, Avro, and Delta formats** using `.write.format().save()`.

- Store each format in different folders (e.g., `/mnt/data/yourname/csv_output/`, `/mnt/data/yourname/parquet_output` etc.).

3. **Compare File Sizes and Read Performance:**
   - Use Databricks `%fs ls /mnt/data/yourname/` to check file sizes.
   - Read each format back into a DataFrame and **measure read times** using Python's `time` module.
   - Compare **compression efficiency** and **read performance**.
4. **Question:**
   - Which format had the smallest file size?
   - Which format had the fastest read time?
   - Why are **columnar formats (Parquet, ORC, Delta)** preferred for analytics?

### Exercise 2: Schema Management in Spark

**Objective:** Understand schema-on-read vs. schema-on-write and practice schema evolution with Delta Lake and Avro.

**Tasks:**

1. **Schema-on-Read vs. Schema-on-Write:**
   - Read a CSV file **without specifying a schema** (`inferSchema=True`).
   - Read the same file **with an explicit schema** (`StructType` in PySpark).
   - Compare the inferred schema vs. explicitly defined schema.
2. **Schema Evolution in Avro:**
   - Write a DataFrame to **Avro format** (`.write.format("avro").save()`).
   - Modify the schema (e.g., add a new column).
   - Read the modified data and explain how Avro handles schema evolution.
3. **Schema Evolution in Delta Lake:**
   - Write the DataFrame as a **Delta table**.
   - Modify the schema (add/remove columns).
   - Use `ALTER TABLE` and `MERGE` to update records while preserving schema changes.
4. **Question:**
   - How does Avro handle schema changes compared to Delta Lake?
   - Why is **schema evolution important** in big data systems?

### Exercise 3: Delta Lake and ACID (Atomicity, Consistency, Isolation and Durability) Transactions

**Objective:** Explore **ACID transactions, time travel**, and **MERGE operations** in **Delta Lake**.

**Tasks:**

1. **Create a Delta Table:**
   - Write a dataset to Delta format (`.write.format("delta").save("/mnt/data/yourname/delta_table")`).
   - Read it back as a **Delta Table**.
2. **Implement ACID Transactions:**
   - Perform **INSERT, UPDATE, and DELETE** operations on the Delta Table using Spark SQL.

3. **Use Time Travel:**
   - Modify the table multiple times.
   - Use `DESCRIBE HISTORY` and `VERSION AS OF` to **query older versions** of the table.
4. **MERGE for Streaming Updates:**
   - Simulate an **incremental update** by creating a new DataFrame with updates.
   - Use **MERGE INTO** to update existing records and insert new ones.
5. **Question:**
   - Why is Delta Lake beneficial for maintaining **data consistency**?
   - How does **time travel** help in debugging data issues?

**Exercise 4: Partitioning and Bucketing**

**Objective:** Optimize queries by **partitioning and bucketing data** to improve performance.

**Tasks:**

1. **Partition Data for Faster Queries:**
   - Write the DataFrame **without partitioning.**
   - Write the same DataFrame **partitioned by a column** (e.g., `year`).
   - Compare query performance using `explain()`.
2. **Bucket Data for Joins:**
   - Bucket a dataset by `customer_id (.bucketBy(10, "customer_id"))`.
   - Perform a join on bucketed and non-bucketed tables.
   - Compare execution plans.
3. **Handle Small Files with Automatic Compaction:**
   - Write small files to a Delta Table.
   - Use `OPTIMIZE` and `ZORDER` to compact and **improve query performance**.
4. **Question:**
   - When should you **partition data?**
   - How does **bucketing improve performance** in joins?
   - What is **Z-order indexing**, and when is it useful?

## Submission Guidelines

- **Email your completed Jupyter Notebook (.ipynb) or Databricks Notebook (.dbc)** so we can review it.
- **Submit early** to allow time for feedback before the next session.
- **Be prepared to share your screen** and explain your work in the next session.
- **Use online resources** if needed, but ensure you understand the concepts.

**Congratulations** on completing the File Formats and Data Storage assignment! This assignment will strengthen your **data storage and optimization skills in Apache Spark**. Looking forward to reviewing your work! Keep exploring and practicing to further solidify your knowledge. Good luck with your continued learning!