Lab 5: Writing a CSV as a Single Output File in DBFS

Author: Dr. Sandeep Kumar Sharma

Learning Objective

In this lab you will learn how to write a Spark DataFrame to DBFS as a single CSV file (one physical file) instead of the default distributed part-files.

Learning Outcome

By the end of this lab, you will be able to: - Understand why Spark writes distributed part-files by default - Use `coalesce()` or `repartition()` to control the number of output files - Save a DataFrame as a single CSV file in DBFS - Optionally move/rename the output part-file to a friendly filename

---

Lab Information

Source CSV (already uploaded to DBFS):

```
/FileStore/tables/employee.csv
```

Target single-file output path (directory):

```
/FileStore/tables/employee_single_output
```

Final friendly filename (optional):

```
/FileStore/tables/employee_single_output/employee_single.csv
```

Step-by-Step Instructions

Step 1 — Read source CSV into a DataFrame

```
input_path = "/FileStore/tables/employee.csv"
df = spark.read.csv(input_path, header=True, inferSchema=True)
display(df)
```

Step 2 — Understand default write behavior (optional check)

Spark will normally write multiple part-files. To see the default behavior, write without coalescing:

```
tmp_output = "/FileStore/tables/employee_output_default"
df.write.mode("overwrite").csv(tmp_output)
# Then list the directory in a notebook cell (dbutils.fs.ls) to see part-files
display(dbutils.fs.ls(tmp_output))
```

Step 3 — Coalesce to a single partition and write

To produce a single physical CSV file, reduce partitions to 1 and write.

```
single_output_dir = "/FileStore/tables/employee_single_output"
# coalesce(1) reduces number of partitions to 1
(df.coalesce(1)
   .write
   .mode("overwrite")
   .option("header", "true")
   .csv(single_output_dir))

# confirm files
display(dbutils.fs.ls(single_output_dir))
```

Step 4 — (Optional) Rename the part file to a friendly filename

Spark will still write a file like `part-00000-...csv`. If you want a friendly filename, move/rename it using dbutils.

```
files = dbutils.fs.ls(single_output_dir)
# find the part file (first file with .csv)
part_file = [f.path for f in files if f.path.endswith('.csv')][0]
final_path = single_output_dir + "/employee_single.csv"
# move (rename)
```

```
dbutils.fs.mv(part_file, final_path)

# list again to confirm
display(dbutils.fs.ls(single_output_dir))
```

Note: `dbutils.fs.mv()` will fail if the target file exists. Remove or overwrite as needed using `dbutils.fs.rm()`.

Step 5 — Read back the single CSV file

```
final_file = "/FileStore/tables/employee_single_output/employee_single.csv"
df_single = spark.read.csv(final_file, header=True, inferSchema=True)
display(df_single)
```

Explanation & Best Practices

- Spark is a distributed engine: by default it writes one file per partition for scalability. When you coalesce to `1`, all data is shuffled (or reduced) to a single executor and written as a single file — this can be expensive for very large datasets.
- Use `coalesce(1)` only for small-to-medium sized outputs or for final export steps.
- If you need deterministic ordering before writing a single file, call `.orderBy(...)` before `coalesce(1)`.
- When working with large production datasets, prefer partitioned parquet/delta formats rather than forcing a single CSV file.

End of Lab 5

You have successfully written a DataFrame as a single CSV file in DBFS and optionally renamed the part-file to a friendly filename.