Lab 6: Reading and Writing Parquet Files in DBFS

Author: Dr. Sandeep Kumar Sharma

Learning Objective

In this lab you will learn how to read from and write to Parquet files using Spark in Azure Databricks, and understand the benefits of Parquet over CSV/JSON for analytics workloads.

Learning Outcome

By the end of this lab, you will be able to: - Read CSV and write as Parquet - Read Parquet files into DataFrames - Understand partitioning and compression options - Use Parquet for efficient querying and storage

---

Lab Information

Source CSV (already available in DBFS):

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

Parquet output directory:

```
/FileStore/tables/employee_parquet
```

Step-by-Step Instructions

Step 1 — Read the source CSV

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

Step 2 — Write DataFrame as Parquet (default options)

```
parquet_output = "/FileStore/tables/employee_parquet"
df.write.mode("overwrite").parquet(parquet_output)
# check files
display(dbutils.fs.ls(parquet_output))
```

Step 3 — Read Parquet File

```
df_parquet = spark.read.parquet(parquet_output)
display(df_parquet)
```

Step 4 — Use partitioning for large datasets (example)

If your data has a natural partition key (for example `department`), you can partition on write to speed up queries that filter by that column.

```
# Example: partition by department
df.write.mode("overwrite").partitionBy("department").parquet(parquet_output +
"_partitioned")
# list partitioned output
display(dbutils.fs.ls(parquet_output + "_partitioned"))
```

Step 5 — Compression options

Parquet supports compression codecs like `snappy`, `gzip`, `brotli`. Snappy is default and fast.

```
# write with snappy compression
(df.write
   .mode("overwrite")
   .option("compression", "snappy")
   .parquet(parquet_output + "_snappy"))
```

Step 6 — Read with explicit schema (optional)

Providing a schema avoids `inferSchema` overhead on large files.

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType

schema = StructType([
    StructField("employee_id", IntegerType(), True),
    StructField("name", StringType(), True),
    StructField("age", IntegerType(), True),
    StructField("department", StringType(), True)
])

# Read CSV with schema and write parquet

df_with_schema = spark.read.csv(input_csv, header=True, schema=schema)
df_with_schema.write.mode("overwrite").parquet(parquet_output + "_with_schema")
```

Explanation & Best Practices

- Parquet is a columnar format which is efficient for analytic queries because it reads only the required columns.
- Parquet stores schema with the data, so downstream readers automatically know column names and types.
- Use partitioning when you have common filtering columns; it reduces the amount of data scanned.
- Use compression (Snappy) for a good balance of speed and storage savings.
- Prefer Parquet or Delta for production analytics instead of CSV for performance and metadata safety.

End of Lab 6

You have learned how to read and write Parquet files in DBFS, use partitioning, and apply compression. Next we can create a Delta lab, or a lab that demonstrates schema evolution and time travel with Delta.