01_Bronze_Ingestion (Python)

Import notebook

Bronze Layer : Raw JSON to delta table

```
import os
from pyspark.sql.functions import col, split, substring_index, sum
from pyspark.sql.types import StructType, StructField, StringType

# --- 1. Configuration ---
# The root of the recreated cvelistV5 structure
root_path = "/Volumes/workspace/default/assignment1/recreated_cvelistV5"

print(f"Starting file count analysis for: {root_path}")

# --- 2. List All Files Recursively ---
# This uses a fast, schema-based read to get all file paths.
file_list_df = (spark.read
    .format("text")
    .schema(StructType([StructField("path", StringType(), True)]))
    .option("recursiveFileLookup", "true")
    .load(os.path.join(root_path, "cves"))
)

# --- 3. Extract Year and Filter for JSON Files ---
# Filter out non-JSON files (like .DS_Store, etc.) and extract the year from the path.
df_with_year = (file_list_df
    .filter(col("path").like("%.json"))
    .withColumn("year", split(col("path"), "/")[7])
)

# --- 4. Count Files Per Year ---
print("\n--- Files Counted Per Year ---")

# Group by the extracted year and count the files.
df_count_by_year = (df_with_year
    .groupBy("year")
    .count()
    .withColumnRenamed("count", "file_count")
    .orderBy(col("year").asc())
)

# Display the results for all years
df_count_by_year.show(df_count_by_year.count(), truncate=False)

# --- 5. Calculate Grand Total ---
# Sum up the 'file_count' column from our aggregated DataFrame.
grand_total = df_count_by_year.agg(sum("file_count")).collect()[0][0]

print("\n--- Overall File Count ---")
print(f"Total JSON files found across all years: {grand_total}")
print("--------------------------")
```

▸ df_count_by_year: pyspark.sql.connect.dataframe.DataFrame = [year: string, file_count: long]
▸ df_with_year: pyspark.sql.connect.dataframe.DataFrame = [path: string, year: string]
▸ file_list_df: pyspark.sql.connect.dataframe.DataFrame = [path: string]

```
Starting file count analysis for: /Volumes/workspace/default/assignment1/recreated_cvelistV5

--- Files Counted Per Year ---
+----+----------+
|year|file_count|
+----+----------+
+----+----------+


--- Overall File Count ---
Total JSON files found across all years: None
--------------------------
```

;

```python
from pyspark.sql.functions import col, year
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, ArrayType, TimestampType

# --- 1. Use complete and correct schema for maximum data quality ---
cve_schema = StructType([
    StructField("dataType", StringType(), True),
    StructField("dataVersion", StringType(), True),
    StructField("cveMetadata", StructType([
        StructField("cveId", StringType(), True),
        StructField("datePublished", StringType(), True),
        StructField("dateUpdated", StringType(), True)
    ])),
    StructField("containers", StructType([
        StructField("cna", StructType([
            StructField("title", StringType(), True),
            StructField("descriptions", ArrayType(StructType([
                StructField("value", StringType(), True)
            ]))),
            StructField("affected", ArrayType(StructType([
                StructField("vendor", StringType(), True),
                StructField("product", StringType(), True)
            ]))),
            StructField("metrics", ArrayType(StructType([
                StructField("cvssV3_1", StructType([
                    StructField("baseScore", DoubleType(), True),
                    StructField("baseSeverity", StringType(), True)
                ]), True),
                StructField("cvssV4_0", StructType([
                    StructField("baseScore", DoubleType(), True),
                    StructField("baseSeverity", StringType(), True)
                ]), True)
            ])))
        ]))
    ]))
])

# --- 2. Configuration and Setup ---
source_path = "/Volumes/workspace/default/assignment1/recreated_cvelistV5/cves"
output_table_name = "cve_bronze.records"
NUM_OUTPUT_FILES = 16

spark.sql("CREATE SCHEMA IF NOT EXISTS cve_bronze")
print(f"Reading all JSON files recursively from: {source_path}")

# --- 3. Run the Ingestion and Filtering Job ---
try:
    print("\nStarting parallel ingestion job...")

    # The full pipeline: Read -> Filter -> Coalesce
    df_optimized = (spark.read
        .schema(cve_schema)
        .option("recursiveFileLookup", "true")
        .json(source_path)
        .filter(col("cveMetadata.cveId").isNotNull())
        .filter(year(col("cveMetadata.datePublished").cast(TimestampType())) == 2024)
        .coalesce(NUM_OUTPUT_FILES)
    )

    print("Data transformation plan is defined. Proceeding to write.")

    # --- 4. Write the DataFrame to the Bronze Delta Table ---
    print(f"\nWriting data to the Delta table: {output_table_name}")
    print("!!! MONITOR THE SPARK UI PROGRESS BAR BELOW THIS CELL !!!")

    (df_optimized.write
        .option("overwriteSchema", "true")
        .mode("overwrite")
        .saveAsTable(output_table_name)
    )

    print("\nBronze layer ingestion complete!")
```

```
        # --- 5. Data Quality Checks and Verification---
        print(f"\n--- Verifying the final table '{output_table_name}' ---")

        final_table = spark.table(output_table_name)
        record_count = final_table.count()
        print(f"SUCCESS: Ingested {record_count} records into {output_table_name}.")

        assert record_count >= 30000, f"DATA QUALITY FAILED: Expected >= 30,000 records, found {record_count}."
        print("✅ Quality Check Passed: Record count is above threshold.")

        print("\n--- Final Bronze Table Sample ---")
        display(final_table.limit(10))

    except Exception as e:
        print(f"AN ERROR OCCURRED: {e}")
```

▸ df_optimized: pyspark.sql.connect.dataframe.DataFrame = [dataType: string, dataVersion: string ... 2 more fields]

▸ final_table: pyspark.sql.connect.dataframe.DataFrame = [dataType: string, dataVersion: string ... 2 more fields]

```
Reading all JSON files recursively from: /Volumes/workspace/default/assignment1/recreated_cvelistV5/cves

Starting parallel ingestion job...
Data transformation plan is defined. Proceeding to write.

Writing data to the Delta table: cve_bronze.records
!!! MONITOR THE SPARK UI PROGRESS BAR BELOW THIS CELL !!!

Bronze layer ingestion complete!

--- Verifying the final table 'cve_bronze.records' ---
SUCCESS: Ingested 40274 records into cve_bronze.records.
✅ Quality Check Passed: Record count is above threshold.

--- Final Bronze Table Sample ---
```

Table

```
    print(record_count)
```

```
40274
```

▸ _sqldf: pyspark.sql.connect.dataframe.DataFrame = [format: string, id: string ... 15 more fields]

Table

This result is stored as `_sqldf` and can be used in other Python and SQL cells.