02_Silver_Normalization (Python)

Import notebook

Silver Layer: Cleaning, Normalization and Validation

```
from pyspark.sql.functions import col, coalesce, explode_outer, to_timestamp

# --- SETUP ---
# Create the target schema for Silver tables if it doesn't already exist.
print("Ensuring schema 'cve_silver' exists...")
spark.sql("CREATE SCHEMA IF NOT EXISTS cve_silver")
print("Schema setup complete.")

# --- LOAD DATA ---
# Load the Bronze table that was created in the bronze notebook.
bronze_df = spark.table("cve_bronze.records")


print("\nSuccessfully loaded and cached the Bronze data.")
display(bronze_df.limit(5))
```

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

```
Ensuring schema 'cve_silver' exists...
Schema setup complete.

Successfully loaded and cached the Bronze data.
```

Table

;

```
# --- 1. Create the Core CVE Table ---

# Select and flatten the necessary fields from the Bronze DataFrame.
silver_cves_df = bronze_df.select(
    col("cveMetadata.cveId").alias("cve_id"),
    # Standardize the date string to a proper timestamp format
    to_timestamp(col("cveMetadata.datePublished")).alias("date_published"),
    col("containers.cna.title").alias("title"),
    # Extract the first description from the descriptions array
    col("containers.cna.descriptions")[0]["value"].alias("description"),
    # Use coalesce to find the first non-null score, searching in order of preference (v4.0, then v3.1)
    coalesce(
        col("containers.cna.metrics")[0]["cvssV4_0"]["baseScore"],
        col("containers.cna.metrics")[0]["cvssV3_1"]["baseScore"]
    ).alias("cvss_score"),
    coalesce(
        col("containers.cna.metrics")[0]["cvssV4_0"]["baseSeverity"],
        col("containers.cna.metrics")[0]["cvssV3_1"]["baseSeverity"]
    ).alias("cvss_severity")
)

# Write the clean, flattened data to a new Silver table.
(silver_cves_df.write
    .mode("overwrite")
    .option("overwriteSchema", "true")
    .saveAsTable("cve_silver.cves")
)

print("Successfully created the 'cve_silver.cves' table.")
display(spark.table("cve_silver.cves"))
```

▸ silver_cves_df: pyspark.sql.connect.dataframe.DataFrame = [cve_id: string, date_published: timestamp ... 4 more fields]

```
Successfully created the 'cve_silver.cves' table.
```

Table

```
# --- 2. Create the Affected Products Table ---

# Explode the 'affected' array to create a one-to-many relationship.
silver_affected_products_df = bronze_df.select(
    col("cveMetadata.cveId").alias("cve_id"),
    explode_outer(col("containers.cna.affected")).alias("affected_product")
).select(
    "cve_id",
    col("affected_product.vendor").alias("vendor"),
    col("affected_product.product").alias("product")
)

# Write the exploded data to its own Silver table.
(silver_affected_products_df.write
    .mode("overwrite")
    .option("overwriteSchema", "true")
    .saveAsTable("cve_silver.affected_products")
)

print("Successfully created the 'cve_silver.affected_products' table.")
display(spark.table("cve_silver.affected_products"))
```

▸ silver_affected_products_df: pyspark.sql.connect.dataframe.DataFrame = [cve_id: string, vendor: string ... 1 more field]

```
Successfully created the 'cve_silver.affected_products' table.
```

Table

```

```

▸ final_silver_table: pyspark.sql.connect.dataframe.DataFrame = [cve_id: string, date_published: timestamp ... 4 more fields]

```
--- Running Data Quality Checks on the 'cve_silver.cves' Table ---
Total 2024 CVEs loaded into Silver table: 40274
✅ Quality Check Passed: Record count is above threshold.
Number of records with null cve_id: 0
✅ Quality Check Passed: No null CVE IDs found.
Total count: 40274 | Distinct CVE IDs: 40274
✅ Quality Check Passed: All CVE IDs are unique.

--- All data quality checks passed! Silver layer is complete. ---
```