

02_Silver_Normalization (Python)[Import notebook](#)

CVE LAKEHOUSE - SILVER LAYER

Silver Layer Configuration

```

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

# Create Silver schema
spark.sql("CREATE SCHEMA IF NOT EXISTS cve_silver")

# Path Configuration
VOLUME_ROOT = "/Volumes/workspace/default/cve_lakehouse_data"
BRONZE_DELTA_PATH = f"{VOLUME_ROOT}/bronze"
SILVER_CORE_PATH = f"{VOLUME_ROOT}/silver/core"
SILVER_AFFECTED_PATH = f"{VOLUME_ROOT}/silver/affected_products"

# Table Names
SILVER_CORE_TABLE = "cve_silver.core"
SILVER_AFFECTED_TABLE = "cve_silver.affected_products"

print("=" * 60)
print("CVE SILVER LAYER - CONFIGURATION")
print("=" * 60)
print(f"Bronze Delta Path : {BRONZE_DELTA_PATH}")
print(f"Silver Core Path : {SILVER_CORE_PATH}")
print(f"Silver Affected Path: {SILVER_AFFECTED_PATH}")
print(f"Core Table Name : {SILVER_CORE_TABLE}")
print(f"Affected Table Name : {SILVER_AFFECTED_TABLE}")
print("=" * 60 + "\n")

# Load Bronze data
print("Loading Bronze data...")
df_bronze = spark.read.format("delta").load(BRONZE_DELTA_PATH)
bronze_count = df_bronze.count()
print(f"✅ Bronze records loaded: {bronze_count:,}\n")

▶ df_bronze: pyspark.sql.connect.DataFrame = [containers: struct, cveMetadata: struct ... 3 more fields]
=====
===== CVE SILVER LAYER - CONFIGURATION =====
===== 
Bronze Delta Path : /Volumes/workspace/default/cve_lakehouse_data/bronze
Silver Core Path : /Volumes/workspace/default/cve_lakehouse_data/silver/core
Silver Affected Path: /Volumes/workspace/default/cve_lakehouse_data/silver/affected_products
Core Table Name : cve_silver.core
Affected Table Name : cve_silver.affected_products
===== 
Loading Bronze data...
✅ Bronze records loaded: 32,924

```

Create Silver Core Table

```
import time
start = time.time()

print("Extracting core CVE fields from Bronze...")

# Explode metrics to get CVSS scores
df_with_metrics = df_bronze.withColumn("metric", explode_outer(col("containers.cna.metrics")))

# Extract core fields
df_core = df_with_metrics.select(
    col("cveMetadata.cveId").alias("cve_id"),
    col("cveMetadata.state").alias("state"),
    col("cveMetadata.datePublished").cast("timestamp").alias("date_published"),
    col("cveMetadata.dateReserved").cast("timestamp").alias("date_reserved"),
    col("cveMetadata.dateUpdated").cast("timestamp").alias("date_updated"),
    coalesce(
        col("metric.cvssV3_1.baseScore"),
        col("metric.cvssV3_0.baseScore"),
        col("metric.cvssV2_0.baseScore")
    ).alias("cvss_base_score"),
    coalesce(
        col("metric.cvssV3_1.baseSeverity"),
        col("metric.cvssV3_0.baseSeverity")
    ).alias("cvss_severity"),
    col("containers.cna.descriptions").getItem(0).getField("value").alias("description")
).dropDuplicates(["cve_id"])

core_count = df_core.count()
print(f"Core records extracted: {core_count:,}")

# Drop existing table
spark.sql("DROP TABLE IF EXISTS cve_silver.core")

# Write directly using saveAsTable
print("\nWriting Silver core table...")
df_core.write.format("delta").mode("overwrite").saveAsTable("cve_silver.core")

elapsed = time.time() - start
print(f"\n✅ Core table created: {core_count:,} records in {elapsed:.2f}s")

# Verify
print("\nSample core records:")
spark.table("cve_silver.core").show(5, truncate=False)
```

```
▶ 📄 df_core: pyspark.sql.connect.dataframe.DataFrame = [cve_id: string, state: string ... 6 more fields]
▶ 📄 df_with_metrics: pyspark.sql.connect.dataframe.DataFrame = [containers: struct, cveMetadata: struct ... 4 more fields]
```

only showing top 5 rows

Core Table Verification and Stats

```
print("=" * 60)
print("SILVER CORE TABLE - VERIFICATION")
print("=" * 60)

# Basic counts
core_count = spark.table("cve_silver.core").count()
print(f"\nTotal Records: {core_count},")

# Show schema
print("\nTable Schema:")
spark.table("cve_silver.core").printSchema()

# Sample records
print("\nSample Records (Key Fields):")
spark.sql("""
    SELECT
        cve_id,
        state,
        date_published,
        cvss_base_score,
        cvss_severity,
        SUBSTRING(description, 1, 80) as description_preview
    FROM cve_silver.core
    LIMIT 5
""").show(truncate=False)

# Data quality stats
print("\nData Quality Statistics:")
spark.sql("""
    SELECT
        COUNT(*) as total_records,
        COUNT(DISTINCT cve_id) as unique_cves,
        COUNT(cvss_base_score) as records_with_cvss,
        SUM(CASE WHEN state = 'PUBLISHED' THEN 1 ELSE 0 END) as published,
        SUM(CASE WHEN state = 'REJECTED' THEN 1 ELSE 0 END) as rejected
    FROM cve_silver.core
""").show(truncate=False)

# CVSS severity distribution
print("\nCVSS Severity Distribution:")
spark.sql("""
    SELECT
        cvss_severity,
        COUNT(*) as count
    FROM cve_silver.core
    WHERE cvss_severity IS NOT NULL
    GROUP BY cvss_severity
    ORDER BY count DESC
""").show(truncate=False)

print("\n✅ Core table verification complete\n")
```

CVSS Severity Distribution:

cvss_severity	count
MEDIUM	10141
HIGH	6670
CRITICAL	1535
LOW	1876
NONE	9

✓ Core table verification complete

Create Silver Affected Products Table

```
import time
start = time.time()

print("Extracting affected products from Bronze...")

# Explode affected products array
df_affected = df_bronze.withColumn("affected", explode_outer(col("containers.cna.affected")))

# Extract vendor and product information
df_products = df_affected.select(
    col("cveMetadata.cveId").alias("cve_id"),
    col("affected.vendor").alias("vendor"),
    col("affected.product").alias("product"),
    col("affected.versions").alias("versions")
).filter(col("vendor").isNotNull() | col("product").isNotNull())

products_count = df_products.count()
print(f"Affected products extracted: {products_count:,}")

# Drop existing table
spark.sql("DROP TABLE IF EXISTS cve_silver.affected_products")

# Write directly using saveAsTable
print("\nWriting Silver affected_products table...")
df_products.write.format("delta").mode("overwrite").saveAsTable("cve_silver.affected_products")

elapsed = time.time() - start
print(f"\nAffected products table created: {products_count:,} records in {elapsed:.2f}s")

# Verify
print("\nSample affected products:")
spark.table("cve_silver.affected_products").show(10, truncate=False)
```

- ▶ df_affected: pyspark.sql.connect.DataFrame = [containers: struct, cveMetadata: struct ... 4 more fields]
- ▶ df_products: pyspark.sql.connect.DataFrame = [cve_id: string, vendor: string ... 2 more fields]

CVE-2024-0217 Red Hat	Red Hat Enterprise Linux 8	NULL
CVE-2024-0217 Red Hat	Red Hat Enterprise Linux 7	NULL
CVE-2024-0217 Red Hat	Red Hat Enterprise Linux 7	NULL

only showing top 10 rows		

Affected Products Table Verification and Stats

```

print("=" * 60)
print("SILVER AFFECTED PRODUCTS TABLE – VERIFICATION")
print("=" * 60)

# Basic counts
products_count = spark.table("cve_silver.affected_products").count()
print(f"\nTotal Records: {products_count},")

# Sample records
print("\nSample Vendor/Product Combinations:")
spark.sql("""
    SELECT
        cve_id,
        vendor,
        product
    FROM cve_silver.affected_products
    WHERE vendor IS NOT NULL
    LIMIT 10
""").show(truncate=False)

# Top vendors
print("\nTop 10 Vendors by Vulnerability Count:")
spark.sql("""
    SELECT
        vendor,
        COUNT(DISTINCT cve_id) as cve_count,
        COUNT(*) as product_count
    FROM cve_silver.affected_products
    WHERE vendor IS NOT NULL
    GROUP BY vendor
    ORDER BY cve_count DESC
    LIMIT 10
""").show(truncate=False)

# Explode effectiveness
print("\nExplode Effectiveness:")
spark.sql("""
    SELECT
        COUNT(DISTINCT cve_id) as unique_cves,
        COUNT(*) as total_vendor_product_records,
        ROUND(COUNT(*) / COUNT(DISTINCT cve_id), 2) as avg_products_per_cve
    FROM cve_silver.affected_products
""").show(truncate=False)

print("\n✅ Affected products table verification complete\n")

```

Apple	468	1492	
Oracle Corporation	366	376	
Cisco	278	363	

Explode Effectiveness:

unique_cves total_vendor_product_records avg_products_per_cve
32574 61255 1.88

Affected products table verification complete

Silver Layer Summary

```

print("=" * 60)
print("SILVER LAYER - FINAL SUMMARY")
print("=" * 60)

# Calculate metrics
core_count = spark.table("cve_silver.core").count()
affected_count = spark.table("cve_silver.affected_products").count()
unique_vendors = spark.sql("SELECT COUNT(DISTINCT vendor) FROM cve_silver.affected_products WHERE vendor IS NOT NULL").collect()[0][0]
unique_cves_in_affected = spark.sql("SELECT COUNT(DISTINCT cve_id) FROM cve_silver.affected_products").collect()[0][0]
avg_products = round(affected_count / unique_cves_in_affected, 1)

# Create summary table with string values
df_summary = spark.createDataFrame([
    ("Core CVE Records", f"{core_count:,}"),
    ("Total Vendor/Product Records", f"{affected_count:,}"),
    ("Unique CVEs in Affected Products", f"{unique_cves_in_affected:,}"),
    ("Unique Vendors", f"{unique_vendors:,}"),
    ("Avg Products per CVE", f"{avg_products:,}")
], ["Metric", "Value"])

print("\nSilver Layer Metrics:")
display(df_summary)

print("\n" + "=" * 60)
print("Silver layer ready for Gold layer analysis!")
print("=" * 60 + "\n")

```

▶ df_summary: pyspark.sql.connect.DataFrame = [Metric: string, Value: string]

```
=====
SILVER LAYER - FINAL SUMMARY
=====
```

Silver Layer Metrics:

Table

```
=====
Silver layer ready for Gold layer analysis!
=====
```

```
● > SyntaxError: invalid decimal literal (command-5345240056320166-3339325405, line 3)
[Trace ID: 00-5f8fc78bf02526bd89ec66d482b1a7dd-8e495a630cc9841c-00]
```