**Case Study: Sales Analytics Platform – Conformed Star Design & SCD-2 Implementation in Spark**

**Scenario:**

A retail company, **RetailX**, operates in multiple countries. It captures:

* Customer purchases across physical stores and online platforms.
* Product returns.
* Regional promotions and seasonal pricing.

The company wants a unified data warehouse where:

* **Sales** and **Returns** are analyzed using shared dimensions.
* **Customer dimension** captures changes in address and loyalty status using **SCD Type-2** logic.
* Data pipelines run daily using **Apache Spark**.

**Part 1: Conformed Star Schema Design**

**1. Identify Business Processes (Facts):**

* Sales
* Returns

**2. Conformed Dimensions (Shared):**

* DimCustomer
* DimProduct
* DimDate
* DimStore
* DimRegion

**3. Bus Matrix:**

| **Business Process** | **Date** | **Customer** | **Product** | **Store** | **Region** |
| --- | --- | --- | --- | --- | --- |
| Sales | Yes | Yes | Yes | Yes | Yes |
| Returns | Yes | Yes | Yes | Yes | Yes |

**4. Star Schema Design (Logical):**

**DimCustomer (SCD-2)**

sql

CustomerSK | CustomerID | Name | Address | LoyaltyTier | StartDate | EndDate | IsCurrent

**DimProduct**

sql

ProductSK | ProductID | Name | Category | Brand

**FactSales**

sql

SalesID | DateSK | CustomerSK | ProductSK | StoreSK | Quantity | Revenue

**FactReturns**

sql

ReturnID | DateSK | CustomerSK | ProductSK | StoreSK | Reason | ReturnAmount

**Part 2: SCD Type-2 Implementation using Apache Spark**

**Goal:**

Maintain history of DimCustomer by inserting a new row when attributes like Address or LoyaltyTier change.

**Step 1: Source Data (Incoming)**

python

source\_df = spark.createDataFrame([

(1, "Alice", "New York", "Gold"),

(2, "Bob", "San Francisco", "Silver"),

(3, "Charlie", "Los Angeles", "Platinum")

], ["CustomerID", "Name", "Address", "LoyaltyTier"])

**Step 2: Load Existing DimCustomer Table**

python

existing\_df = spark.read.format("delta").load("path/to/dim\_customer")

# Sample schema of existing\_df

# CustomerSK | CustomerID | Name | Address | LoyaltyTier | StartDate | EndDate | IsCurrent

**Step 3: Join & Compare to Detect Changes**

python

from pyspark.sql.functions import col, lit, current\_date

# Join on business key

joined\_df = source\_df.alias("src").join(

existing\_df.filter("IsCurrent = true").alias("tgt"),

on="CustomerID",

how="left"

)

# Find rows where dimension data changed

changed\_df = joined\_df.filter(

(col("src.Address") != col("tgt.Address")) |

(col("src.LoyaltyTier") != col("tgt.LoyaltyTier"))

).select("src.\*")

**Step 4: Mark Existing Rows as Historical**

python

from pyspark.sql.functions import expr

expired\_df = joined\_df.filter(

(col("src.Address") != col("tgt.Address")) |

(col("src.LoyaltyTier") != col("tgt.LoyaltyTier"))

).select("tgt.CustomerSK").withColumn("EndDate", current\_date()) \

.withColumn("IsCurrent", lit(False))

**Step 5: Insert New Versioned Rows**

python

from pyspark.sql.functions import monotonically\_increasing\_id

new\_version\_df = changed\_df \

.withColumn("CustomerSK", monotonically\_increasing\_id()) \

.withColumn("StartDate", current\_date()) \

.withColumn("EndDate", lit(None).cast("date")) \

.withColumn("IsCurrent", lit(True))

**Step 6: Combine & Write Back**

python

final\_df = existing\_df \

.filter("IsCurrent = true") \

.join(expired\_df, "CustomerSK", "left\_anti") \

.unionByName(expired\_df) \

.unionByName(new\_version\_df)

final\_df.write.format("delta").mode("overwrite").save("path/to/dim\_customer")

**Optional Enhancements:**

* Add CDC timestamps from operational system.
* Create a function to automate SCD-2 handling for any dimension.
* Implement deduplication logic for high-frequency updates.

**Outcome:**

* You now have a **conformed dimensional model** for Sales and Returns.
* The DimCustomer dimension uses **Spark Delta** to support SCD Type 2.
* This design ensures consistent analytics across multiple fact tables and supports historical tracking.