Given the problem scenario, our goal is to design a system to efficiently manage customer data from multiple countries, allowing us to process billions of records daily. We need to consider the following key components:

1. **Source System Data Format**: The data file contains customer records with both header (H) and detail (D) records. Each customer has multiple details across countries, but we will need to capture the latest consultation record for each customer based on the date of consultation.
2. **Splitting Data by Country**: We need to dynamically split the data by country and load it into country-specific tables (e.g., Table\_India, Table\_USA). This requires us to know the relevant country from the data itself, which is provided in the Country field.
3. **Handling Shift of Customers Between Countries**: If a customer moves from one country to another and consults in the new country, we must ensure the data with the latest consultation date is kept. Thus, for each customer, we will keep only the latest record based on the Last\_Consulted\_Date.
4. **Derived Columns**:
   * **Age**: Calculate the age of the customer based on their DOB.
   * **Days Since Last Consulted**: Calculate the number of days since the last consultation (Last\_Consulted\_Date) compared to the current date. If this is greater than 30 days, flag it.

**Steps for the Solution:**

1. **Create Country-Specific Tables**:
   * Each country will have a separate table to store the customer data.
   * These tables will have additional derived columns such as Age and Days\_Since\_Last\_Consulted.
2. **Create the Table Schema**:
   * For each country, we need the following columns:
     + Customer\_Name, Customer\_Id, Open\_Date, Last\_Consulted\_Date, Vaccination\_Id, Dr\_Name, State, Country, DOB, Is\_Active, Age, Days\_Since\_Last\_Consulted.
3. **Data Processing Flow**:
   * **Extract and Transform**:
     + Parse the incoming data file.
     + Filter only the D records (Details records).
     + Extract customer data and update derived columns.
     + If the customer already exists in the country-specific table, compare the Last\_Consulted\_Date and keep the latest one.
   * **Validation**:
     + Ensure data integrity (e.g., correct date formats, no missing mandatory fields like Customer\_Id or Country).
     + Validate that the data for each customer in each country is consistent and correct.
4. **Handling Large Volumes of Data**:
   * Use **distributed computing frameworks** (like **PySpark**) to handle large datasets efficiently.
   * Process data in chunks or batches to avoid memory overload and optimize performance.

Table schema

CREATE TABLE Table\_<Country> (

Customer\_Name STRING,

Customer\_Id STRING PRIMARY KEY,

Open\_Date DATE,

Last\_Consulted\_Date DATE,

Vaccination\_Id STRING,

Dr\_Name STRING,

State STRING,

Country STRING,

DOB DATE,

Is\_Active CHAR(1),

Age INT,

Days\_Since\_Last\_Consulted INT

);

**Derived Columns:**

1. **Age**:
   * Age = DATEDIFF(CURRENT\_DATE, DOB) / 365 (rounded to the nearest integer).
2. **Days Since Last Consulted**:
   * Days\_Since\_Last\_Consulted = DATEDIFF(CURRENT\_DATE, Last\_Consulted\_Date).
3. **Consultation Date Validation**:
   * Ensure the Last\_Consulted\_Date is not in the future and is a valid date format.

**Python/PySpark Implementation**

We'll use PySpark for handling large-scale data processing. Below is a basic outline of the solution:

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, datediff, current\_date, to\_date, when

# Initialize Spark Session

spark = SparkSession.builder.appName("CustomerDataProcessing").getOrCreate()

# Function to process data from a file

def process\_customer\_data(file\_path):

# Read the data into a DataFrame

df = spark.read.option("delimiter", "|").csv(file\_path, header=False, inferSchema=True)

# Assign column names based on header

df = df.withColumnRenamed("\_c0", "Record\_Type") \

.withColumnRenamed("\_c1", "Customer\_Name") \

.withColumnRenamed("\_c2", "Customer\_Id") \

.withColumnRenamed("\_c3", "Open\_Date") \

.withColumnRenamed("\_c4", "Last\_Consulted\_Date") \

.withColumnRenamed("\_c5", "Vaccination\_Id") \

.withColumnRenamed("\_c6", "Dr\_Name") \

.withColumnRenamed("\_c7", "State") \

.withColumnRenamed("\_c8", "Country") \

.withColumnRenamed("\_c9", "DOB") \

.withColumnRenamed("\_c10", "Is\_Active")

# Filter out header records (Record\_Type = 'H')

df\_details = df.filter(col("Record\_Type") == "D")

# Parse dates

df\_details = df\_details.withColumn("Open\_Date", to\_date(col("Open\_Date"), "yyyyMMdd")) \

.withColumn("Last\_Consulted\_Date", to\_date(col("Last\_Consulted\_Date"), "yyyyMMdd")) \

.withColumn("DOB", to\_date(col("DOB"), "yyyyMMdd"))

# Add derived columns: Age and Days Since Last Consulted

df\_details = df\_details.withColumn("Age", (datediff(current\_date(), col("DOB")) / 365).cast("int"))

df\_details = df\_details.withColumn("Days\_Since\_Last\_Consulted", datediff(current\_date(), col("Last\_Consulted\_Date")))

# Validate: Ensure no future consultation dates

df\_details = df\_details.withColumn("Is\_Valid\_Consultation", when(col("Last\_Consulted\_Date") <= current\_date(), True).otherwise(False))

# Filter out invalid records (future consultations)

df\_details = df\_details.filter(col("Is\_Valid\_Consultation") == True)

# Now, process data by country

countries = df\_details.select("Country").distinct().collect()

for country\_row in countries:

country = country\_row["Country"]

country\_df = df\_details.filter(col("Country") == country)

# If there are existing records for the country, merge them based on the latest consultation date

# Assuming you are working with a SQL-based system (could be Spark SQL, PostgreSQL, etc.)

# Create or update the corresponding country-specific table (e.g., Table\_USA, Table\_IND)

country\_df.createOrReplaceTempView(f"temp\_{country}")

spark.sql(f"""

INSERT INTO Table\_{country}

SELECT Customer\_Name, Customer\_Id, Open\_Date, Last\_Consulted\_Date, Vaccination\_Id,

Dr\_Name, State, Country, DOB, Is\_Active, Age, Days\_Since\_Last\_Consulted

FROM temp\_{country}

WHERE Last\_Consulted\_Date = (SELECT MAX(Last\_Consulted\_Date) FROM temp\_{country} WHERE Customer\_Id = temp\_{country}.Customer\_Id)

""")

return "Data Processing Completed."

# Example of processing a file

process\_customer\_data("path\_to\_data\_file.csv")

**Explanation:**

1. **Reading the Data**: We read the input file using Spark’s CSV reader, specifying | as the delimiter.
2. **Renaming Columns**: We rename the columns based on the provided header format.
3. **Filtering and Parsing**: We filter out the header rows and parse date columns (Open\_Date, Last\_Consulted\_Date, DOB).
4. **Adding Derived Columns**: We calculate the Age and Days\_Since\_Last\_Consulted columns using PySpark's built-in functions.
5. **Validating Records**: We ensure the Last\_Consulted\_Date is not a future date.
6. **Processing by Country**: We filter the records by Country, and for each country, we check if the latest consultation date for a given Customer\_Id is already in the table. If not, we insert it into the corresponding country table.

**Handling Scale:**

* **Distributed Processing**: PySpark allows us to handle large volumes of data in parallel, processing records across multiple nodes.
* **Efficient Storage**: We use country-specific tables to reduce data duplication and make the querying process more efficient.

This solution should scale to handle billions of records efficiently. If we move to a real-world scenario, we would likely use Spark on a distributed cluster, leveraging HDFS or a similar distributed storage system to handle the vast amount of data.