PharmaSUG 2026 - Paper AI-103

Data Without Borders: CDISC Data Hub for Multi-Language Clinical Analytics & AI

Mayank Singh, J&J MedTech (Neurovascular)

ABSTRACT

In the evolving landscape of clinical research, fragmented data environments hinder rapid insights, cross- study analysis, and regulatory compliance. This paper introduces a scalable and flexible approach for centralized clinical data repositories, designed to be agnostic to underlying relational database systems— our implementation leverages Amazon (AWS) Redshift. The framework employs structured SDTM schemas and a dynamic approach for ADaM. Automated, language-agnostic ETL (Extraction,Transformation, Loading) pipelines facilitate seamless data access across SAS, R, Python, and SQL, supporting advanced analytics, machine learning, and meta-analyses. By transforming traditional study- specific storage into an integrated ecosystem, this framework addresses data inconsistency and silos, promotes collaboration among multidisciplinary teams, and ensures compliance with industry standards. The proposed solution empowers clinical organizations to accelerate scientific discovery, foster innovation, and adapt to evolving data standards—paving the way for a future of truly borderless clinical data analytics.

INTRODUCTION

The clinical research domain is characterized by an ever-increasing volume of complex data generated from multiple studies, sources, and formats. Traditionally, data is stored in study-specific folders, which leads to several challenges: data silos, inconsistency, duplication, and difficulty in cross-study analysis. These issues impede rapid insights, compromise data integrity, and hinder regulatory compliance.

Modern clinical organizations require scalable, flexible, and efficient data architectures to support advanced analytics, machine learning, AI applications, and real-time decision-making. Data engineering—the process of designing, building, and maintaining data pipelines and repositories—is at the core of this transformation. It involves not only data storage but also data integration, cleaning, transformation, and providing seamless multi-language access to facilitate diverse analytical workflows.

This paper proposes a comprehensive approach designed to be agnostic to underlying relational database systems (AWS Redshift in our case), tailored for clinical datasets like SDTM and ADaM. The architecture leverages scalable, flexible schema designs and automated pipelines to enable efficient data access, transformation, and analytics, supporting the modern needs of healthcare, pharma, and medical device industries.

MOTIVATION AND CHALLENGES

The increasing complexity and volume of clinical and healthcare data demand innovative data engineering solutions that can support rapid insights, regulatory compliance, and technological advancements. As organizations aim to leverage advanced analytics, machine learning, and AI to accelerate drug/device development, improve patient outcomes, and ensure regulatory compliance, there

is a clear need for a modern, scalable, and flexible data infrastructure.

Furthermore, the heterogeneity of data sources and formats necessitates a unified approach to data management that supports diverse analytical tools and programming languages. The ability to automate data pipelines, support evolving schemas, and facilitate multi-language access is crucial for enabling rapid, reliable insights. This motivation underscores the importance of adopting a comprehensive data engineering strategy that transforms fragmented data environments into integrated, scalable ecosystems capable of supporting innovation and compliance in a fast-paced industry.

CHALLENGES WITH TRADITIONAL DATA STORAGE

- **Fragmentation:** Study-specific folders lead to data duplication, inconsistent formats, and difficulty in aggregating data.
- **Limited Cross-Study Analysis:** Isolated data hampers meta-analyses, comparative studies, and machine learning model training.
- **Manual Data Handling:** Time-consuming extraction, transformation, and loading (ETL) processes increase risk of errors and delays.
- **Regulatory & Compliance Risks:** Disparate data sources complicate audit trails and regulatory submissions.
- **Limited Flexibility:** Rigid schemas and siloed storage inhibit rapid adaptation to evolving analysis needs.

PROPOSED ARCHITECTURE & METHODOLOGY

PREREQUISITES

To use this approach, users should have the following Authentications along with SAS 9.4:

- **Amazon Redshift:** Users must request IAM roles or access keys from the AWS Service Administrator to securely connect and query the data warehouse.
- **SAS:** SAS should be licensed with the ODBC (SAS/ACCESS Interface to ODBC) connector to enable seamless connectivity to Redshift. ODBC stands for Open Database Connectivity.
- **Redshift Driver:** The Amazon Redshift ODBC driver must be properly configured within SAS to facilitate reliable data access and transfer.

OBJECTIVE

The primary objectives of this approach are:

- Centralize data storage in a scalable cloud environment.
- Support flexible schema designs.
- Automate ETL pipelines for reliable, repeatable data processing.
- Enable multi-language access for diverse analytical tools.
- Support advanced analytics, ML, and AI applications at scale.
- Be adaptable to different database systems and evolving industry standards.

OVERVIEW OF ARCHITECTURE

The proposed architecture leverages a centralized clinical data repository built on AWS Redshift, integrated with multiple analytical and visualization tools. The architecture comprises three core components:

- **AWS Redshift Data Warehouse:** Serves as the scalable, secure, and unified storage platform for SDTM and ADaM datasets, supporting flexible schema designs and high-performance querying.
- **Programming & Analytics Platforms:** Includes SAS, R, Python, and SQL environments that connect to Redshift via configured drivers (e.g., ODBC), enabling multi-language data access, advanced analytics, machine learning, and dashboarding.
- **ETL & Data Processing Pipelines:** Automated data extraction, transformation, and loading processes developed using SAS and R scripts (with the flexibility to implement in other languages), which process raw data from source systems and populate the Redshift repository.

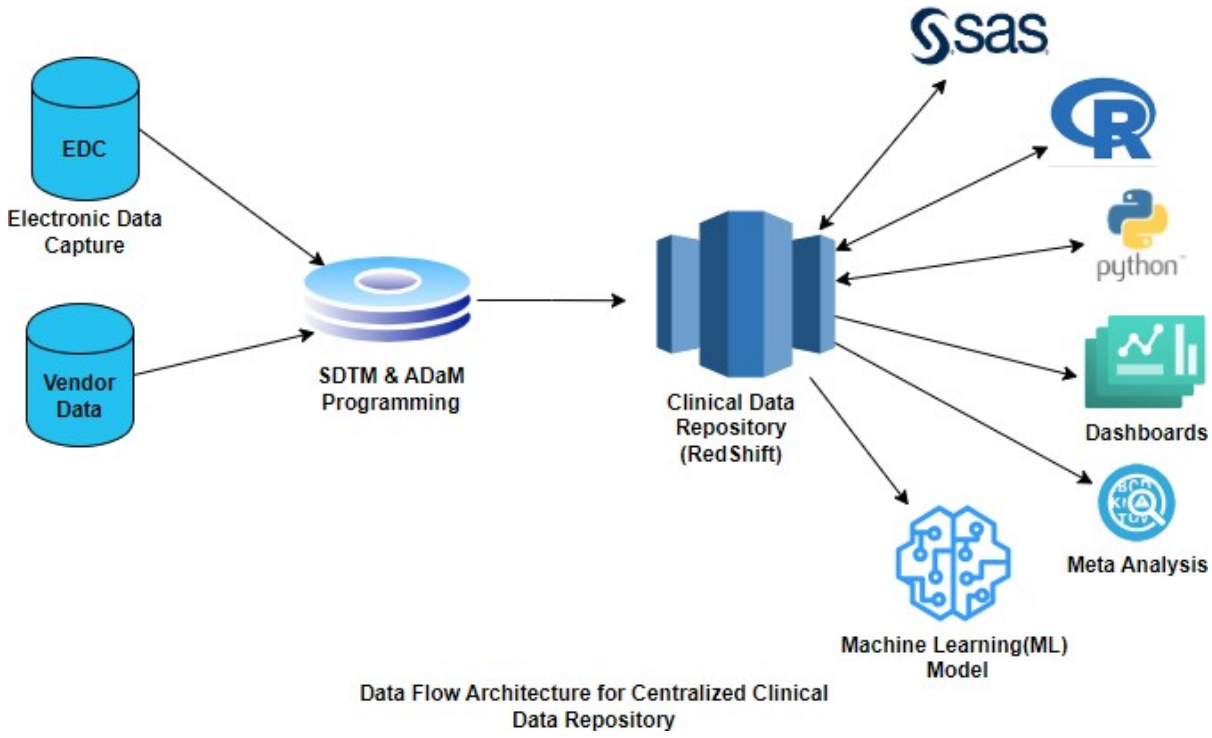


Figure 1. Data Flow Architecture

DATA STORAGE LAYER

The foundation of architecture involves designing flexible and scalable schemas for SDTM and ADaM datasets:

- **AWS Redshift:** Acts as a scalable, secure, and centralized data warehouse for highly structured SDTM, less – structured ADaM and various non-CDISC data sources.
- **SDTM Data:** Structured according to CDISC standards (in our case IGv3.3), stored in domain-specific tables with predefined schemas to ensure compliance and consistency.
    - Metadata is utilized to define and generate tables for SDTM domains. This metadata includes all potential variables within each domain, which depend on the event class, in accordance with the CDISC SDTM model (version 1.7 in our implementation).
    - Each domain can encompass data from multiple studies; specific study can be selected through a 'where' clause for targeted querying.

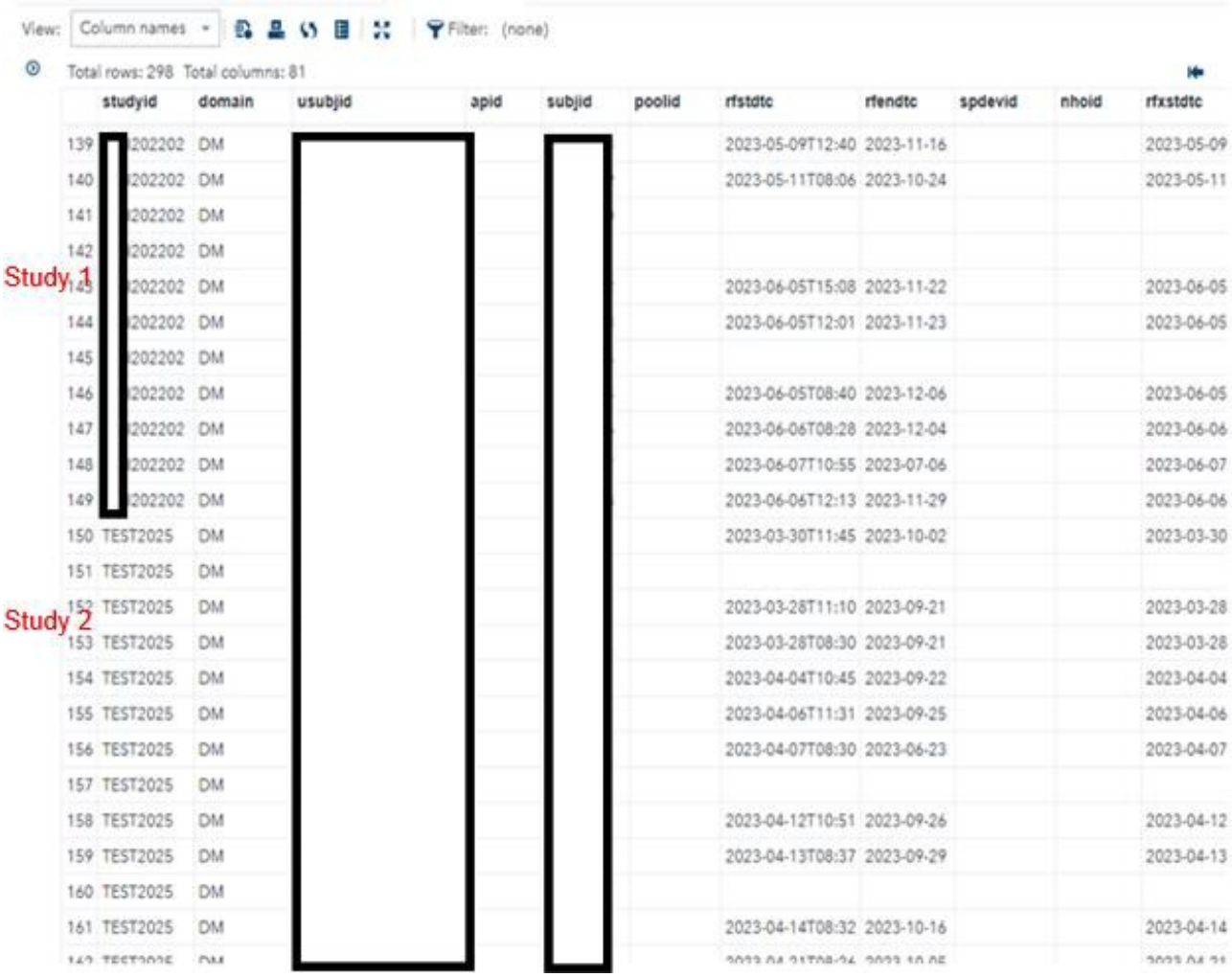
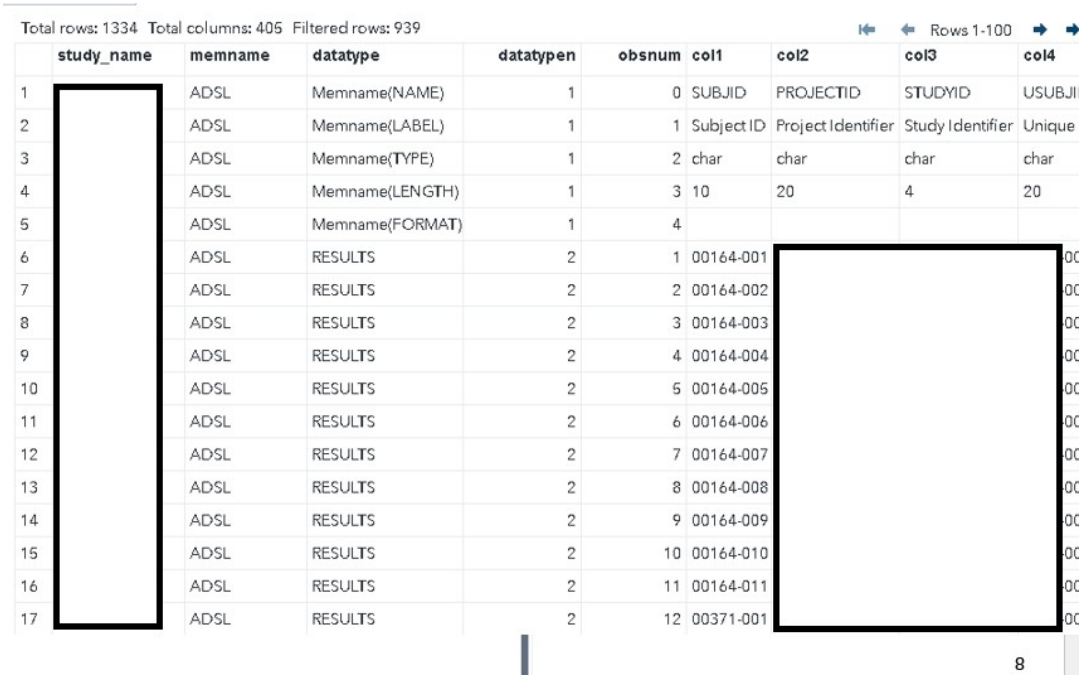


Figure 2. Illustrates SDTM stored in RedShift DM Table

- **ADaM Data:** Stored using a **generic column approach**— Instead of creating multiple schemas/tables for different datasets and for different studies, each data point is stored as a record with metadata describing its nature. This approach is used because ADaM structure has lot of variability across studies.

  This flexibility allows us to:

  - Supports multiple data types and structures without schema modifications
  - Allows dynamic ingestion of new datasets
  - Simplifies data integration from various sources
  - Reduces complexity and maintenance overhead



Total rows: 1334 Total columns: 405 Filtered rows: 939 Rows 1-100

	study_name	memname	datatype	datatypen	obsnum	col1	col2	col3	col4
1		ADSL	Memname(NAME)	1	0	SUBJID	PROJECTID	STUDYID	USUBJI
2		ADSL	Memname(LABEL)	1	1	Subject ID	Project Identifier	Study Identifier	Unique
3		ADSL	Memname(TYPE)	1	2	char	char	char	char
4		ADSL	Memname(LENGTH)	1	3	10	20	4	20
5		ADSL	Memname(FORMAT)	1	4				
6		ADSL	RESULTS	2	1	00164-001			
7		ADSL	RESULTS	2	2	00164-002			
8		ADSL	RESULTS	2	3	00164-003			
9		ADSL	RESULTS	2	4	00164-004			
10		ADSL	RESULTS	2	5	00164-005			
11		ADSL	RESULTS	2	6	00164-006			
12		ADSL	RESULTS	2	7	00164-007			
13		ADSL	RESULTS	2	8	00164-008			
14		ADSL	RESULTS	2	9	00164-009			
15		ADSL	RESULTS	2	10	00164-010			
16		ADSL	RESULTS	2	11	00164-011			
17		ADSL	RESULTS	2	12	00371-001			

8

Figure 3. Illustrates ADaM stored in RedShift Table

Below is an example of how SDTM structures are created in RedShift using SAS.

```
/*************************************************************************************
  Macro Name: SDTM Schema Builder
  Author: Mayank Singh
  Purpose: Create SDTM tables in Redshift DB
*************************************************************************************/

/* Define AWS Redshift User ID as a macro variable */
%let aws_rs_uid = <YOUR_AWS_Redshift_UID>;

/* Define AWS Redshift Password as a macro variable */
%let aws_rs_pass = <YOUR_AWS_Redshift_PASS>;

/* Define AWS Server (hostname or IP) */
%let aws_ser = <YOUR_AWS_Server>;

/* Define AWS Database name */
%let aws_db = <YOUR_AWS_Database>;
```

```
/* Define AWS Schema name where tables will be created */
%let aws_sch = <YOUR_AWS_Schema>;

/* Define AWS Port number for connection */
%let aws_prt = <YOUR_AWS_Port>;

/* Import the Excel file containing SDTM structure metadata into a SAS dataset */
Proc import datafile="/Standards/REDSHIFT_DB_SDTM_STRUCT.xlsx"
out=REDSHIFT_META dbms=xlsx replace;
run;

/* Data step to process the imported metadata and prepare SQL statements */
data REDSHIFT_META;
length R_TYPE $20. M_VAR L_VAR $150.; /* Define variable lengths */
set REDSHIFT_META;

/* Determine data type based on 'DB_Type' column */
if DB_Type='Num' then R_TYPE="Int"; /* Numeric type */
else R_TYPE="VARCHAR"||"("||strip(DB_Length)||")"; /* Varchar with length */

/* Prepare variable name for SQL, wrapping in quotes */
VARIABLE_='"'||strip(Variable)||'"';

/* Concatenate variable name and type for table creation */
M_VAR=catx(' ',Variable_,R_TYPE);

/* Prepare SQL comment statement for each column */
L_VAR="COMMENT ON COLUMN &aws_sch..SDTM_"||strip(domain)||"."||strip(Variable)||" IS"||"
'"||strip(label)||"';";

run;

/* Macro to create SDTM tables in Redshift, optionally deleting existing tables */
%macro Create_DB_STRUCT(Delete_Table=);

  /* Count the number of unique domains in the metadata */
  Proc sql;
  select count(distinct domain) into:ndom trimmed from REDSHIFT_META;

  /* Get list of distinct domains into macro variables dom1 - dom&ndom */
  select distinct domain into:dom1 -:dom&ndom. from REDSHIFT_META;
  quit;

  /* Loop through each domain to create corresponding table */
  %do i=1 %to &ndom.;

    /* Retrieve variable definitions for current domain, ordered by Variable_ORD */
    Proc sql;
    select distinct M_VAR into:DBVAR separated by ',' from REDSHIFT_META
    where Domain="&&dom&i."
    order by Variable_ORD;

    /* Retrieve column comments for current domain, ordered similarly */
    select distinct L_VAR into:DBVARL separated by " " from REDSHIFT_META
    where Domain="&&dom&i."
    order by Variable_ORD;
    quit;


    /* Establish ODBC connection to Redshift with the specified credentials */
    libname RSHIFT odbc noprompt="Driver={Amazon Redshift (x64)}; Server=&aws_ser;
      Database=&aws_db.; UID=&aws_rs_uid.; PWD=&aws_rs_pass.; Port=&aws_prt" schema=&aws_SCH.;

    /* Connect to Redshift using PROC SQL and ODBC connection */
    proc sql;
    connect to odbc as myredshift_conn (noprompt="Driver={Amazon Redshift (x64)};
Server=&aws_ser;
      Database=&aws_db.; UID=&aws_rs_uid.; PWD=&aws_rs_pass.; Port=&aws_prt; ");

    /* Optionally drop existing table if Delete_Table=Y */
    %if &Delete_Table.=Y %then %do;
```

```
      EXECUTE (DROP TABLE IF EXISTS &aws_sch.sdtm_&&dom&i.) BY myredshift_conn;
    %end;

    /* Create new table with the defined variables if it doesn't exist */
    EXECUTE (
      create table IF NOT EXISTS &aws_sch.sdtm_&&dom&i. (
        &DBVAR. /* Variable definitions with types */
      );
    ) BY myredshift_conn;

    /* Add comments to each column in the table */
    EXECUTE (
      &DBVARL. /* Column comment statements */
    ) BY myredshift_conn;

    /* Disconnect from Redshift */
    DISCONNECT FROM myredshift_conn;
    quit;

  %end; /* End of loop over domains */
%mend; /* End of macro */

%Create_DB_STRUCT(Delete_Table=Y);
```

Program 1. SDTM Schema Creation using SAS

DATA PIPELINES & PROCESSING

- **ETL Development:** Automated pipelines built using SAS and R scripts, following data engineering principles such as modularity, idempotency, and scalability.
- **Language-Agnostic Pipelines:** Designed to be extensible, enabling implementation in other languages like Python, Java, or SQL stored procedures, ensuring maximum flexibility.

Below is an example of how the ADaM data is extracted to users with original structures using R.

```
# Program Name: Extract ADaM
# Author: Mayank Singh
# Purpose: Extract ADaM data from Redshift DB

# Load necessary libraries
library(DBI)
library(dplyr)
library(tidyr)
library(stringr)
library(glue)
library(RPostgreSQL)

# Define the function
ads_extract <- function(studyid, out) {

  # Establish connection to Redshift (use environment variables or secure method in
production)
  con <- dbConnect(dbDriver("PostgreSQL"),
                   dbname = "xxx",
                   host = "XXX",
                   port =  XXXX,
                   user = "XXX",
                   password = "XXX")

  on.exit(dbDisconnect(con))

  # Fetch total distinct memnames for the study
  query_nds <- glue("SELECT COUNT(DISTINCT memname) AS nds FROM nv_workspace.ads_data
WHERE study_name = '{studyid}'")
```

```
  nds_result <- dbGetQuery(con, query_nds)
  nds <- nds_result$nds[1]

  # Fetch list of memnames
  query_ds <- glue("SELECT DISTINCT memname FROM nv_workspace.ads_data WHERE study_name =
'{studyid}'")
  ds_names_df <- dbGetQuery(con, query_ds)
  ds_names <- ds_names_df$memname
  message("Memnames found: ", paste(ds_names, collapse = ", "))

  # Loop over each memname
  for (mem in ds_names) {
    # Convert to lowercase for case-insensitive matching
    study_lower <- tolower(studyid)
    mem_lower <- tolower(mem)

    # Fetch data for the current memname
    sql_query <- glue(
      "SELECT * FROM nv_workspace.ads_data WHERE LOWER(study_name) = '{study_lower}' AND
LOWER(memname) = '{mem_lower}'"
    )
    df <- dbGetQuery(con, sql_query)

    # Separate meta-data and results data
    meta_df <- df %>% filter(datatype != 'RESULTS')
    results_df <- df %>% filter(datatype == 'RESULTS')

    # Define number of columns dynamically or set statically if known
    ncol <- 400
    cols_to_transpose <- paste0("col", 1:ncol)

    # Helper function to pivot data
    pivot_metadata <- function(datatype_value, new_col_name) {
      meta_df %>%
        filter(datatype == datatype_value) %>%
        pivot_longer(
          cols = all_of(cols_to_transpose),
          names_to = "var",
          values_to = new_col_name
        )
    }

    # Pivot for each metadata attribute
    name_df <- pivot_metadata('Memname(NAME)', 'A1')
    label_df <- pivot_metadata('Memname(LABEL)', 'A2')
    type_df <- pivot_metadata('Memname(TYPE)', 'A3')
    length_df <- pivot_metadata('Memname(LENGTH)', 'A4')
    format_df <- pivot_metadata('Memname(FORMAT)', 'A5')

    # Combine all metadata attributes
    meta_combined <- name_df %>%
      left_join(label_df, by = "var") %>%
      left_join(type_df, by = "var") %>%
      left_join(length_df, by = "var") %>%
      left_join(format_df, by = "var") %>%
      select(study_name.x = study_name.x, memname.x = memname.x, var, A1, A2, A3, A4, A5)

    # Filter valid metadata rows
    df_metadata <- meta_combined %>%
      filter(A1 != '', A2 != '') %>%
      mutate(A3 = if_else(A3 == 'num' & A5 != '', A5, A3))



    # Keep only variables that are in metadata
    keep_vars <- df_metadata$var
    df_data <- results_df %>% select(all_of(keep_vars))
```

```
    # Loop through each metadata row to assign new variables with attributes
    for (i in seq_len(nrow(df_metadata))) {
      old_var <- df_metadata$var[i]
      new_var <- df_metadata$A1[i]
      attr_type <- df_metadata$A3[i]
      attr_label <- df_metadata$A2[i]
      attr_length <- df_metadata$A4[i]

      if (old_var %in% names(df_data)) {
        # Copy old variable to new variable
        df_data[[new_var]] <- df_data[[old_var]]

        # Coerce type if specified
        if (!is.null(attr_type) && attr_type != "") {
          if (attr_type == "num") {
            df_data[[new_var]] <- as.numeric(df_data[[new_var]])
          } else if (attr_type == "char") {
            df_data[[new_var]] <- as.character(df_data[[new_var]])
          } else if (attr_type == "integer") {
            df_data[[new_var]] <- as.integer(df_data[[new_var]])
          } else if (attr_type == "DATE9.") {

            df_data[[new_var]] <- as.Date(df_data[[new_var]], format="%d%b%Y")


          } else if (grepl("DATETIME", attr_type)) {
            df_data[[new_var]] <- as.POSIXct(df_data[[new_var]], format="%d%b%y:%H:%M",
tz="UTC")
            #as.Date(df_data[[new_var]], format="%d%b%y:%H:%M")
          }
          # Add other types if necessary
        }

        # Assign label attribute
        if (!is.null(attr_label) && attr_label != "") {
          attr(df_data[[new_var]], "label") <- attr_label
        }

        # Assign length attribute
        if (!is.null(attr_length) && attr_length != "") {
          attr(df_data[[new_var]], "length") <- attr_length
        }
      }
    }

    # Remove original variables, keep only new ones
    df_data <- df_data %>% select(!all_of(df_metadata$var))

    # Save output with unique filename
    out_file <- paste0(out, "_", mem)
    assign(out_file, df_data, envir = .GlobalEnv)
  }

  message("Extraction complete.")
}

# Usage:
ads_extract("ABC", "st1")
```

Program 2. ADaM Data extraction using R

	SUBJID Subject ID	PROJECTID Project Identifier	STUDYID Study Identifier	USUBJID Unique Subject Identifier	REGION Region	SITEID Study Site Identifier	SITENAME Site Name
1	00934-013	[redacted]	6321	[redacted]-00934-013	US	4235	934 - [redacted]
2	00934-014		6321	-00934-014	US	4235	934 -
3	00934-016		6321	-00934-016	US	4235	934 -
4	00934-017		6321	-00934-017	US	4235	934 -
5	00934-018		6321	-00934-018	US	4235	934 -
6	00934-019		6321	-00934-019	US	4235	934 -
7	00934-020		6321	-00934-020	US	4235	934 -
8	00934-021		6321	-00934-021	US	4235	934 -
9	00934-022		6321	-00934-022	US	4235	934 -
0	00934-023		6321	-00934-023	US	4235	934 -
1	00934-025		6321	-00934-025	US	4235	934 -
2	00934-026		6321	-00934-026	US	4235	934 -
3	00934-027		6321	-00934-027	US	4235	934 -
4	00934-029		6321	-00934-029	US	4235	934 -
5	00934-030		6321	-00934-030	US	4235	934 -
6	00934-031		6321	-00934-031	US	4235	934 -
7	00934-032		6321	-00934-032	US	4235	934 -
8	00934-033		6321	-00934-033	US	4235	934 -
9	01150-005		6321	-01150-005	US	4747	0115
0	01150-006		6321	-01150-006	US	4747	0115
1	00934-034		6321	-00934-034	US	4235	934 -

Figure 4. Illustrates ADaM Extracted from RedShift

Multi-Environment Data Access & Analytics

- **Multi-Language Connectivity:** Data is accessible via SAS, R, Python, and SQL, supporting diverse analytical workflows.
- **Secure Connectivity:** JDBC/ODBC drivers, API endpoints, and secure authentication mechanisms facilitate high-performance, compliant data access.
- **Advanced Analytics & AI:** Integration with cloud services like AWS SageMaker, QuickSight, and external tools for machine learning, meta-analyses, and visualization.

BENEFITS

- **Scalability & Performance:** Cloud infrastructure supports large-scale data storage and high-performance querying.
- **Data Consistency:** Centralized repository reduces discrepancies and ensures data integrity.
- **Multi-Language Support:** Facilitates diverse workflows, enabling statisticians, data scientists, and analysts to work in their preferred environments.

- **Advanced Analytics & AI:** Supports machine learning, AI applications, and meta-analyses on integrated datasets, accelerating insights.
- **Automation & Reliability:** Automated pipelines reduce manual effort, errors, and delays.
- **Flexibility & Extensibility:** Schema design and pipeline architecture support evolving standards, data types, and analytical needs.

CONCLUSION

This paper presents a comprehensive, scalable architecture that transforms traditional fragmented clinical data environments into a unified, multi-language clinical analytics hub. By leveraging relational database (AWS Redshift in our case), structured SDTM schemas, and a flexible ADaM approach, coupled with automated ETL pipelines, our solution enables seamless data access, integration, and advanced analytics across diverse tools such as SAS, R, Python, and SQL. This architecture not only addresses critical challenges of data inconsistency and silos but also fosters collaboration, accelerates insights, and ensures regulatory compliance. Looking ahead, this framework offers a versatile blueprint adaptable to evolving data standards and emerging technologies, empowering clinical organizations to harness the full potential of their data assets and drive innovation in a borderless, data-driven future.

REFERENCES

- CDISC Standards
- AWS Redshift Documentation
- Industry Best Practices for Data Warehousing
- Regulatory Guidelines: FDA, EMA, and other regulatory bodies
- SAS Documentation:
- R Documentation:

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Mayank Singh

Johnson & Johnson MedTech (Neurovascular)

msing133@its.jnj.com