PharmaSUG 2026 - Paper DV-383

Closing the Loop: An Interactive R Shiny Dashboard for EDC Data Visualization and Real-Time Review Tracking

Jun Yang and Yuying Jin, Organon LLC

ABSTRACT

Clinical data review often involves a fragmented workflow where medical monitors and data managers switch between EDC systems, static PDF listings, and Excel trackers. This creates silos between the data and the review process, making it difficult to track query status in real-time. There is a critical need for a centralized platform that combines data visualization with a formal "review and comment" loop.

This paper introduces a "Full-Stack" R Shiny application designed to bridge the gap between raw EDC data and structured data review. The application utilizes the R Shiny dashboard framework to provide a multi-tab interface for loading raw datasets (CSV/SAS7BDAT). Interactive visualizations, including patient profiles and safety trend plots, are generated using Plotly and ggplot2. The core innovation is the integration of the DT (DataTables) package with a **MySQL database backend**. By leveraging DBI and pool packages, the app allows users to select specific rows, mark them as "Reviewed," and enter comments directly within the UI. These entries are immediately persisted to a MySQL table, ensuring a single, auditable source of truth for the review status.

Implementing this dashboard significantly reduces the time from data entry to insight. By automating the link between visualization and documentation, study teams can identify outliers faster and maintain a transparent audit trail of the review process. The session will conclude with a demonstration of the "Write-Back" logic and database schema considerations for clinical environments.

INTRODUCTION

R Shiny is an open-source web application framework that enables users to build interactive web applications and dashboards entirely in the R (or Python) programming language, without requiring traditional web development skills like HTML, CSS, or JavaScript.

MySQL is a widely used, open-source relational database management system (RDBMS) that uses Structured Query Language (SQL) for managing and querying data. It is known for its reliability, speed, and scalability, powering many of the world's largest websites and applications.

This project includes user interface design, database design, data integrity validation, data manipulation, and various types of charts and plots. Specifically, we will demonstrate the development of an R Shiny web application for reviewing and editing clinical data, combining an excel source with a MySQL database. The solution is implemented using the open-source shiny, DT, DBT/RMariaDB, openxlsx, dplyr libraries.

PRIOR WORK

Various approaches have been proposed for developing tools to analyze data using languages like SAS, Java and Python. However, regarding cost-effectiveness, capability, superior data visualization and strong community and collaboration, R is generally considered “better” than Python and SAS specific contexts such as clinical trial specific plots and listing tables.

WALKTHROUGH

PRELIMINARIES

We need to set up the environment for R and MySQL and install necessary packages.

Environment:

1. **R base**: https://cran.r-project.org/bin/windows/base/
2. **R studio**: https://posit.co/download/rstudio-desktop/
3. **MySQL database**: https://www.mysql.com/downloads/

After installing MySQL database, please memorize username and password for later use. Also, we suggest to install MySQL Workbench which is a unified visual tool for database architects, developers, and DBAs.

R Package:

1. **shiny** - web app framework
2. **DT** — interactive data tables
3. **DBI / RMariaDB** — MySQL database connectivity
4. **openxlsx** — reading Excel files
5. **dplyr** — data manipulation

To install the packages above, please use the following command in the console at R studio.

```
> install.packages(c("shiny","DT","DBI","RMariaDB","openxlsx","dplyr"))
```

Below is the program skeleton. We will show the rest of code throughout the walkthrough:

```
library(shiny)
library(DT)
library(DBI)
library(RMariaDB)
library(openxlsx)
library(dplyr)

ui <- fluidPage(

)

server <- function(input, output, session) {


}

shinyApp(ui, server)
```

UI CODE

This object controls the layout, appearance, and placement of all the visual elements in the web application, such as text, widgets (sliders, buttons, etc.), and where outputs (plots, tables) will appear. It defines the "frontend" of the app. We can define listing table, drop down list and different plots at UI section.

```
ui <- fluidPage(
  selectInput(
    "status_filter",
    "Filter by Status:",
    choices = c("All", "pending", "reviewed")
  ),
  DTOutput("table", width = "80%"),
  actionButton("save", "Save Changes")
)
```

```
selectInput(
    "status_filter",
    "Filter by Status:",
    choices = c("All", "pending", "reviewed")
  ),
```

A dropdown to filter records by status (All, pending, reviewed).

```
DTOutput("table", width = "80%"),
```

An interactive table displaying the data with customized 80% width.

```
actionButton("save", "Save Changes")
```

A Save Changes button to persist edits.

SERVER LOGIC

This is a function that contains the logic and instructions for the app. It uses the input from the UI to perform R computations and determines how to update the output elements in the UI. The server function is re-run each time a new user visits the app, ensuring each user has an isolated session.

The server section includes 4 parts working on different purposes: data loading, reactivity, cell editing and saving. Reactivity and cell editing are complex as R Shiny utilizes web programming techniques to catch and process events. In the following section, we demonstrate to leverage these capabilities to load data from an excel file and MySQL database, catch any operations and update parts of application when the user changes input values, and push back records into MySQL database.

```r
server <- function(input, output, session) {

  # Create connection
  con <- dbConnect(
    RMariaDB::MariaDB(),
    dbname = "r_shiny_dashboard",
    host = "127.0.0.1",
    user = "root",
    password = "MySQLTest",
    port = 3306
  )

  # Ensure connection closes when app stops
  onStop(function() {
    dbDisconnect(con)
  })

  excel_data <- read.xlsx("D:/PharamSUG_2026/R/Data/Demo.xlsx", sheet = "sd_explus")

  query <- "select record_id, domain,status, comments from review_result LIMIT 100"
  mysql_data <- dbGetQuery(con, query)

  username <- Sys.info()[["user"]]

  data_combined <- excel_data %>%
    left_join(mysql_data, by = "record_id")

  rv <- reactiveValues(
    data = data_combined,
    changed = data.frame()   # store only edited rows
  )

  filtered_data <- reactive({

    df <- rv$data

    if (input$status_filter != "All") {
      df <- df[df$status == input$status_filter, ]
    }

    df
  })


  output$table <- renderDT({
    datatable(
      filtered_data(),
      editable = list(
        target = "cell",
        disable = list(columns = c(0, 1))  # disable Excel columns
      ),
      options = list(pageLength = 10)
```

```
  )
})

observeEvent(input$table_cell_edit, {
  info <- input$table_cell_edit
  i <- info$row
  j <- info$col
  v <- info$value

  colname <- names(filtered_data())[j]

  # Restrict status values
  if (colname == "status") {
    if (!(v %in% c("pending", "reviewed"))) {
      showNotification("Invalid value! Use 'pending' or 'reviewed'", type = "error")
      return()
    }
  }

  # Update main data
  rv$data[i, j] <- v

  # Extract the full row
  updated_row <- rv$data[i, ]

  # Remove duplicates (same record_id)
  rv$changed <- rv$changed[rv$changed$record_id != updated_row$record_id, ]

  # Add updated row
  rv$changed <- rbind(rv$changed, updated_row)
})

observeEvent(input$save, {

  df <- rv$changed

  for (i in 1:nrow(df)) {

    record_id <- df$record_id[i]

    # Check if exists
    exists <- dbGetQuery(
      con,
      sprintf("SELECT COUNT(*) as n FROM review_result WHERE record_id = '%s'", record_id)
    )$n > 0

    if (exists) {
      # UPDATE
      dbExecute(con, sprintf(
        "UPDATE review_result SET status='%s',comments='%s',modified_by='%s' WHERE record_id='%s'",
        df$status[i],
        df$comments[i],
        username,
        record_id
      ))
    } else {
      # INSERT
      dbExecute(con, sprintf(
        "INSERT INTO review_result (record_id, status, comments,subject_Id, created_by)
         VALUES ('%s', '%s', '%s', '%s','%s')",
        record_id,
        df$status[i],
        df$comments[i],
        df$Subject.Identifier.for.the.Study[i],
        username
      ))
    }
  }

  showNotification("Data saved successfully!", type = "message")
})
}
```

```
  # Create connection
  con <- dbConnect(
    RMariaDB::MariaDB(),
    dbname = "r_shiny_dashboard",
    host = "127.0.0.1",
    user = "root",
    password = "MySQLTest",
    port = 3306
  )
```

Create a database connection and put necessary information like credential and database name and port number.

```
  # Ensure connection closes when app stops
  onStop(function() {
    dbDisconnect(con)
  })
```

Connecting to a database is used some resources. After done, we need to disconnect to the database.

```
excel_data <- read.xlsx("data/Data/Demo.xlsx", sheet = "sd_explus")
```
Read data on the specific tab on the excel file into the data frame. There is another package used to read SAS format data.

```
 query <- "select record_id, domain,status, comments from review_result LIMIT 100"
 mysql_data <- dbGetQuery(con, query)
```
Define SQL query and up to 100 records. Then execute the SQL query to pull data from the database into the data frame.

```
username <- Sys.info()[["user"]]
```
We need to log who creates and updates records as the part of log information.

```
data_combined <- excel_data %>%
    left_join(mysql_data, by = "record_id")
```
Merge both clinical data with reviewing result data by the primary key.

```
  rv <- reactiveValues(
    data = data_combined,
    changed = data.frame()   # store only edited rows
  )
```
Creates a list-like object used to store multiple variables.

```
  filtered_data <- reactive({
      df <- rv$data
      if (input$status_filter != "All") {
      df <- df[df$status == input$status_filter, ]
    }
      df
  })
```
Perform data manipulation using predefined filter to get specific group data.

```
  output$table <- renderDT({
    datatable(
      filtered_data(),
      editable = list(
        target = "cell",
        disable = list(columns = c(0, 21))  # disable Excel columns
      ),
      options = list(pageLength = 10)
    )
  })
```
Render an interactive data table in R Shiny and set the table editable in which only variables from database can be editable.

```
observeEvent(input$table_cell_edit, {
    info <- input$table_cell_edit
    i <- info$row
    j <- info$col
    v <- info$value

    colname <- names(filtered_data())[j]

    # Restrict status values
    if (colname == "status") {
      if (!(v %in% c("pending", "reviewed"))) {
        showNotification("Invalid value! Use 'pending' or 'reviewed'", type = "error")
        return()
      }
    }

    # Update main data
```

```
    rv$data[i, j] <- v

    # Extract the full row
    updated_row <- rv$data[i, ]

    # Remove duplicates (same record_id)
    rv$changed <- rv$changed[rv$changed$record_id != updated_row$record_id, ]

    # Add updated row
    rv$changed <- rbind(rv$changed, updated_row)
  })
```

Use observe Event function to react to cell edits in the listing table. It catches which row and column is edited and the value of the cell. As we want to allow users to type in only two choices: pending, reviewed, so the validation function to guarantee it. Otherwise, the warning message will pop up. Also, we only insert and update records which are never existing in the database, so keep updating the list.

```
 observeEvent(input$save, {

    df <- rv$changed

    for (i in 1:nrow(df)) {

      record_id <- df$record_id[i]

      # Check if exists
      exists <- dbGetQuery(
        con,
        sprintf("SELECT COUNT(*) as n FROM review_result WHERE record_id = '%s'",
record_id)
      )$n > 0

      if (exists) {
        # UPDATE
        dbExecute(con, sprintf(
          "UPDATE review_result SET status='%s',comments='%s',modified_by='%s' WHERE
record_id='%s'",
          df$status[i],
          df$comments[i],
          username,
          record_id
        ))
      } else {
        # INSERT
        dbExecute(con, sprintf(
          "INSERT INTO review_result (record_id, status, comments,subject_Id, created_by)
           VALUES ('%s', '%s', '%s', '%s','%s')",
          record_id,
          df$status[i],
          df$comments[i],
          df$Subject.Identifier.for.the.Study[i],
          username
        ))
      }
    }

    showNotification("Data saved successfully!", type = "message")
  })
```

Upon clicking the Save button, the code loops through changed rows and performs insert or update records based on whether the records are existing in the database or not. If record_id exists in the DB, then update status, comments, and modified_by (current OS user). If not, then insert a new record including subject_id and created_by. After done, displays a success notification on completion.

RUN

After coded UI components and server components, we need to call the specific core command to R Shiny package to create and run an interactive web application. It takes two main components as arguments: ui (user interface) and server (computational logic).

```
shinyApp(ui, server)
```

INTERACTIVE LISTING TABLE

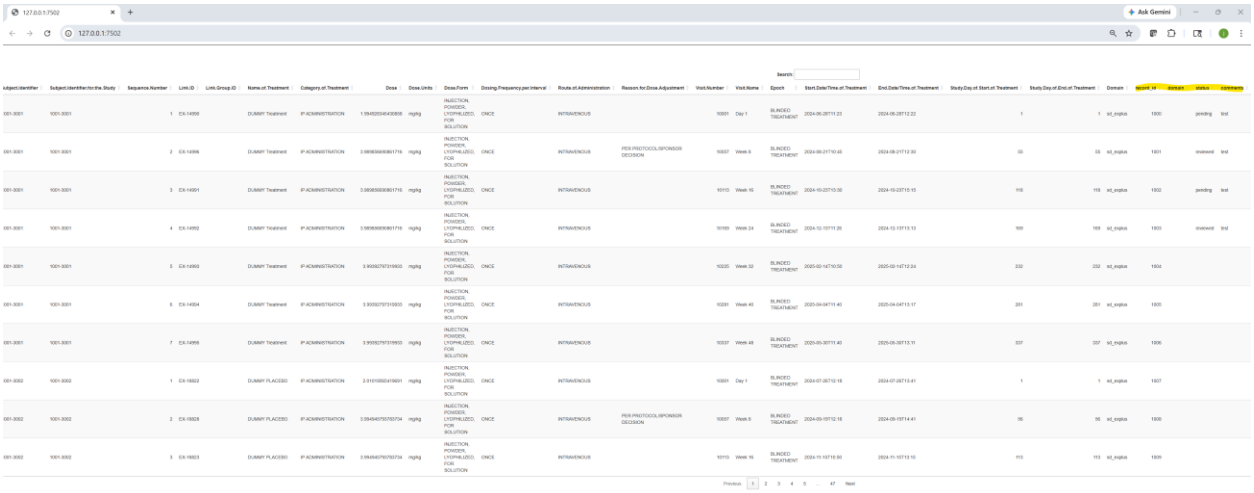

Figure 1: An example of the interactive listing table in which the highlighted variables come from MySQL database.

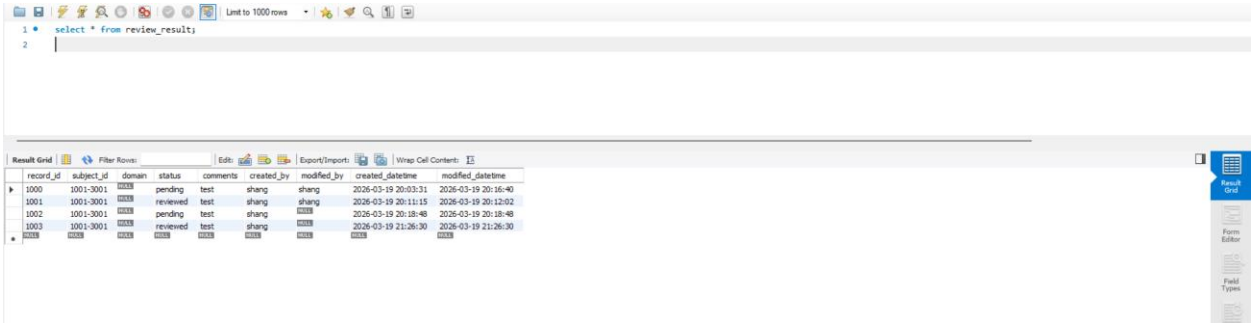

Figure 2: An example of data in MySQL database.

ACTUAL OUPUT

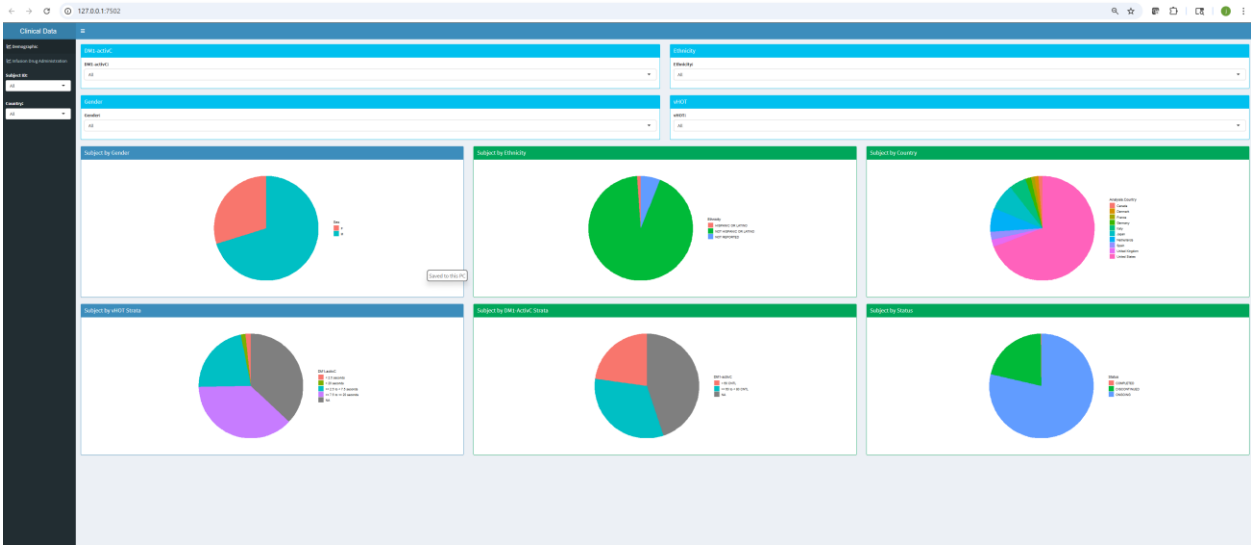

Figure 3: An example of dashboard with multiple tabs and different filters and charts.

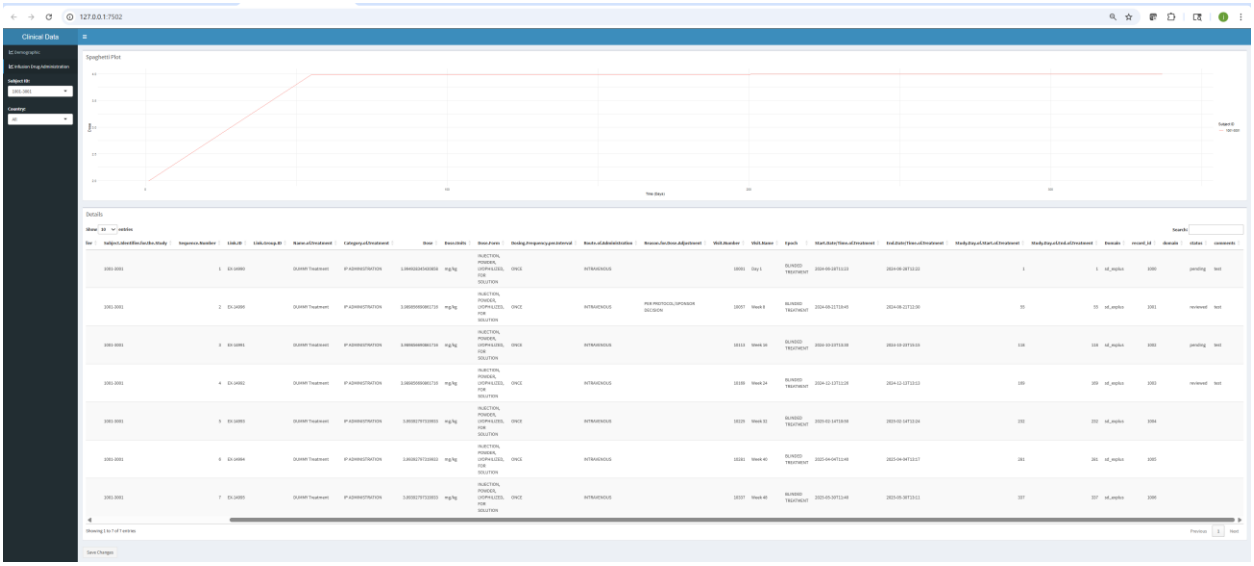

Figure 4: An example of listing table and line plot filtered by a subject ID.

CONCLUSION

We have demonstrated the web application developed using R Shiny and a MySQL database. The web application is capable of not only creating different plots using ggplot package, but also communicating with MySQL database to persist review results for clinical data. In fact, dynamic and interactive dashboard can be both straightforward and user-friendly through all available modes: **automation**, **customization and auditability**, all made possible using free and open-source libraries.

We hope the information provided empowers you to develop tailored solutions that meet your specific needs.

REFERENCES

[1] MySQL https://www.mysql.com/

[2] R Shiny https://shiny.posit.co/

[3] R DT https://rstudio.github.io/DT/

[4] R RMariaDB https://mariadb.com/docs/connectors/other/rmariadb

[5] R openxlsx https://ycphs.github.io/openxlsx/index.html

[6] R dplyr https://dplyr.tidyverse.org/

CONTACT INFORMATION

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

Jun Yang
Organon LLC
jun.yang@organon.com

Yuying Jin
Organon LLC
yuying.jin@organon.com