Shiny application for UI to `workout.duckdb` Database

J. Christopher Westland

2025-03-03

On Blackboard, I have posted a Shiny application that provides a user-friendly interface for updating each field in each table within the `workout.duckdb` database.

Features:

1. **Table Selection**: Users can select a table to edit.
2. **Record Editing**: Users can view, update, and delete records.
3. **New Record Addition**: Users can insert new records.
4. **Data Fetching**: Users can refresh and view table data in real-time.
5. **REST API Communication**: Uses your existing Plumber API for database operations.

Shiny UI Code

This code creates the Shiny interface, fetching data from your DuckDB via the server API, and providing CRUD operations.

```r
library(shiny)
library(httr)
```

```
Warning: package 'httr' was built under R version 4.4.2
```

```r
library(jsonlite)
```

```
Warning: package 'jsonlite' was built under R version 4.4.2


Attaching package: 'jsonlite'

The following object is masked from 'package:shiny':

    validate
```

```
library(DT)
```

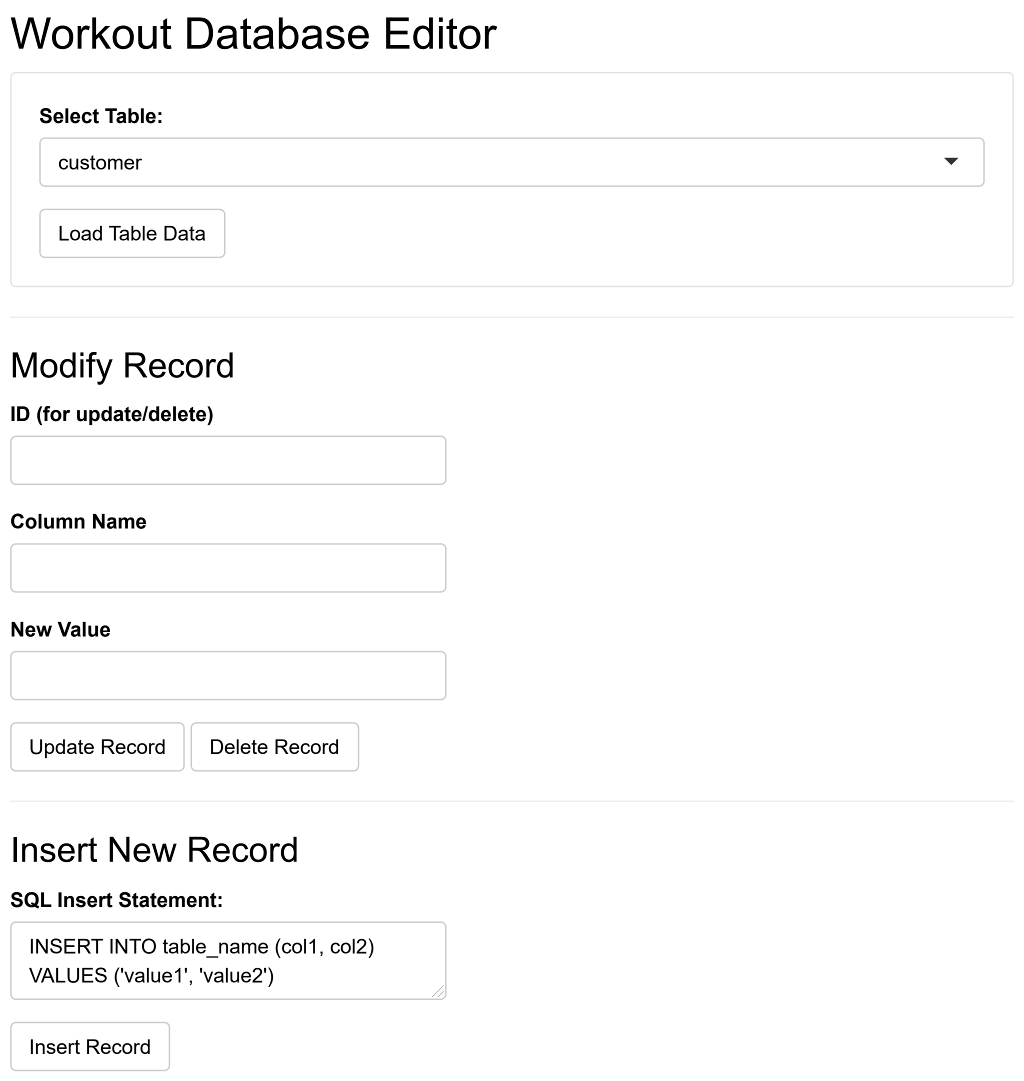
```

Attaching package: 'DT'

The following objects are masked from 'package:shiny':

    dataTableOutput, renderDataTable
```

```
# Define API base URL
base_url <- "http://localhost:8000"

# UI
ui <- fluidPage(
  titlePanel("Workout Database Editor"),

  sidebarLayout(
    sidebarPanel(
      selectInput("table_name", "Select Table:", choices = NULL),
      actionButton("load_data", "Load Table Data"),
      br(),
      textOutput("status_message")
    ),

    mainPanel(
      DTOutput("table_data"),

      hr(),
      h3("Modify Record"),
      fluidRow(
        column(3, textInput("record_id", "ID (for update/delete)", "")),
        column(3, textInput("update_column", "Column Name", "")),
        column(3, textInput("update_value", "New Value", ""))
      ),
      actionButton("update_record", "Update Record"),
      actionButton("delete_record", "Delete Record"),

      hr(),
      h3("Insert New Record"),
      textAreaInput("insert_sql", "SQL Insert Statement:",
                    "INSERT INTO table_name (col1, col2) VALUES ('value1', 'value2')"),
      actionButton("insert_record", "Insert Record")
    )
  )
)

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

  # Load available tables from DuckDB
  observe({
    res <- GET(url = paste0(base_url, "/query"), query = list(sql = "SHOW TABLES"))
    tables <- fromJSON(content(res, as = "text", encoding = "UTF-8"))

    if (!"error" %in% names(tables)) {
      updateSelectInput(session, "table_name", choices = tables$name)
    } else {
      output$status_message <- renderText("Error fetching tables!")
    }
  })

  # Load data from the selected table
  observeEvent(input$load_data, {
    req(input$table_name)
    query_sql <- paste("SELECT * FROM", input$table_name)
    res <- GET(url = paste0(base_url, "/query"), query = list(sql = query_sql))
    data <- fromJSON(content(res, as = "text", encoding = "UTF-8"))

    if (!"error" %in% names(data)) {
      output$table_data <- renderDT(data, editable = TRUE)
    } else {
      output$status_message <- renderText("Error loading data!")
    }
```

```
})

# Update a record
observeEvent(input$update_record, {
  req(input$table_name, input$record_id, input$update_column, input$update_value)
  # Check if table is sales_invoice, and use invoice_id instead of id
  primary_key <- ifelse(input$table_name == "sales_invoice", "invoice_id", "id")

  update_sql <- paste0("UPDATE ", input$table_name, " SET ", input$update_column,
                       " = '", input$update_value, "' WHERE ", primary_key, " = ", input$re

  res <- POST(url = paste0(base_url, "/execute"), body = list(sql = update_sql), encode = "
  result <- fromJSON(content(res, as = "text", encoding = "UTF-8"))

  if (!"error" %in% names(result)) {
    output$status_message <- renderText("Record updated successfully!")
  } else {
    output$status_message <- renderText(result$error)
  }
})

# Delete a record
observeEvent(input$delete_record, {
  req(input$table_name, input$record_id)
  delete_sql <- paste0("DELETE FROM ", input$table_name, " WHERE id = ", input$record_id)
  res <- POST(url = paste0(base_url, "/execute"), body = list(sql = delete_sql), encode = "
  result <- fromJSON(content(res, as = "text", encoding = "UTF-8"))

  if (!"error" %in% names(result)) {
    output$status_message <- renderText("Record deleted successfully!")
  } else {
    output$status_message <- renderText(result$error)
  }
})

# Insert a new record
observeEvent(input$insert_record, {
  req(input$insert_sql)
  res <- POST(url = paste0(base_url, "/execute"), body = list(sql = input$insert_sql), enco
  result <- fromJSON(content(res, as = "text", encoding = "UTF-8"))

  if (!"error" %in% names(result)) {
```

```r
      output$status_message <- renderText("Record inserted successfully!")
    } else {
      output$status_message <- renderText(result$error)
    }
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```

```
Listening on http://127.0.0.1:6754
```



```
INSERT INTO table_name (col1, col2)
VALUES ('value1', 'value2')
```

How It Works

1. **Table Selection**: The app fetches all tables from the database.
2. **Data Display**: Clicking “Load Table Data” fetches and displays the data.
3. **Record Modification**:
   - **Update**: Users specify a column and new value, which updates a record based on its ID.
   - **Delete**: Deletes a record using its ID.
   - **Insert**: Accepts an SQL `INSERT` statement to add a new record.

This Shiny app enables full database interaction using your existing client-server architecture. Let me know if you need adjustments!