Exercise IV – R Shiny

Part 4 – Connect to Database

CEE412 / CET522

TRANSPORTATION DATA MANAGEMENT AND VISUALIZATION

WINTER 2020

Introduction

- In the previous exercise you built a user interface from scratch.
- This exercise is about creating a user interface that visualizes data accessed from a SQL database.
- Additionally, in this exercise we will introduce code about building a survey for a Shiny application interface and communicating the entered survey data back to a SQL database.

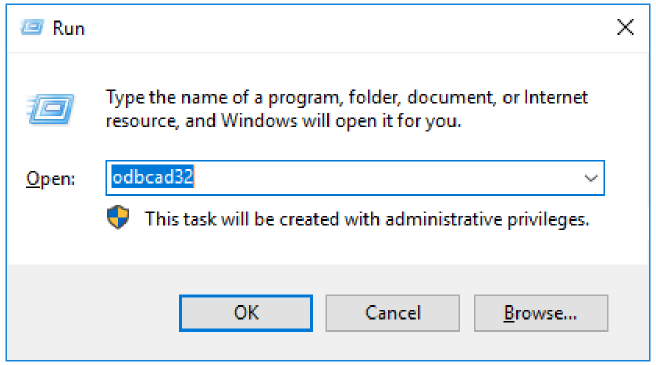
Connect to Database

- We have learned how to use RODBC to connect to database and manipulate data. Shiny also support RODBC and other database related packages.

- We will show them by using several examples:
  1. Creating DSN and using RODBC to query and visualize data in a Shiny App
  2. Collect data from Shiny App and save data into SQL Server
  3. Visualize updated data in a Shiny App

Step1: Create a DSN

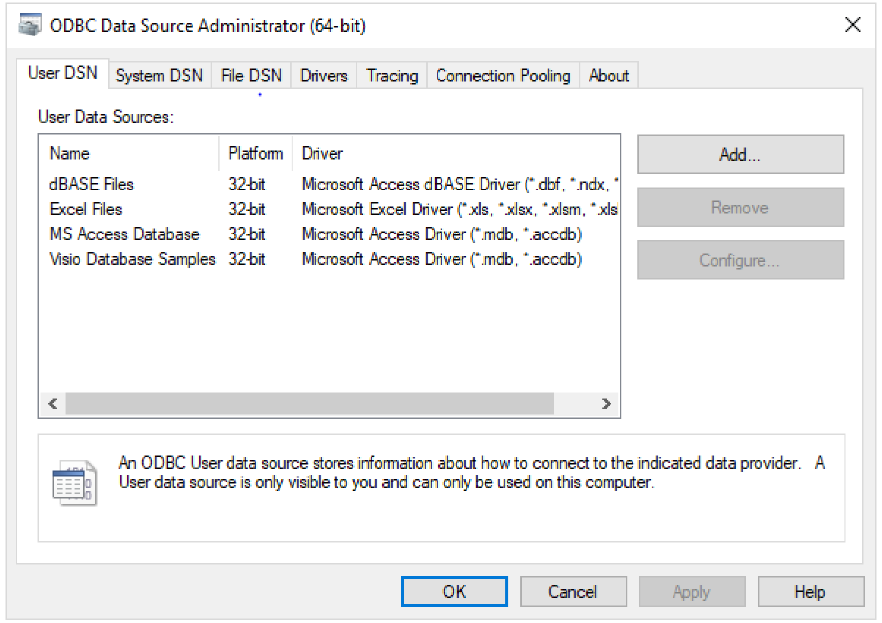
- Click **Start → Run** or Win+R and type "odbcad32"
- Click **Add** to add a new DSN, select SQL Server and click **Finish**.


Run

Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: odbcad32

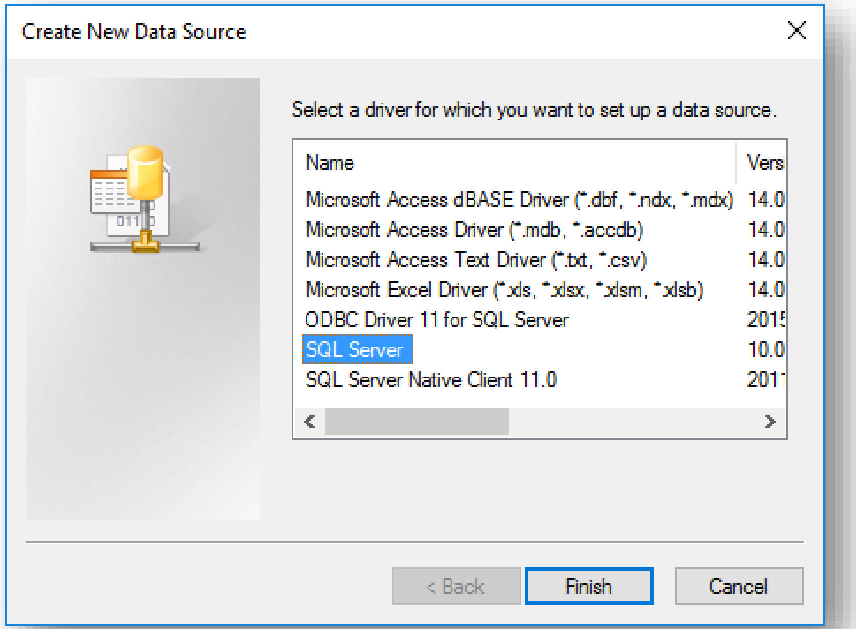
This task will be created with administrative privileges.


ODBC Data Source Administrator (64-bit)

User Data Sources:

Name	Platform	Driver
dBASE Files	32-bit	Microsoft Access dBASE Driver (*.dbf, *.ndx, *
Excel Files	32-bit	Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xls
MS Access Database	32-bit	Microsoft Access Driver (*.mdb, *.accdb)
Visio Database Samples	32-bit	Microsoft Access Driver (*.mdb, *.accdb)

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you and can only be used on this computer.

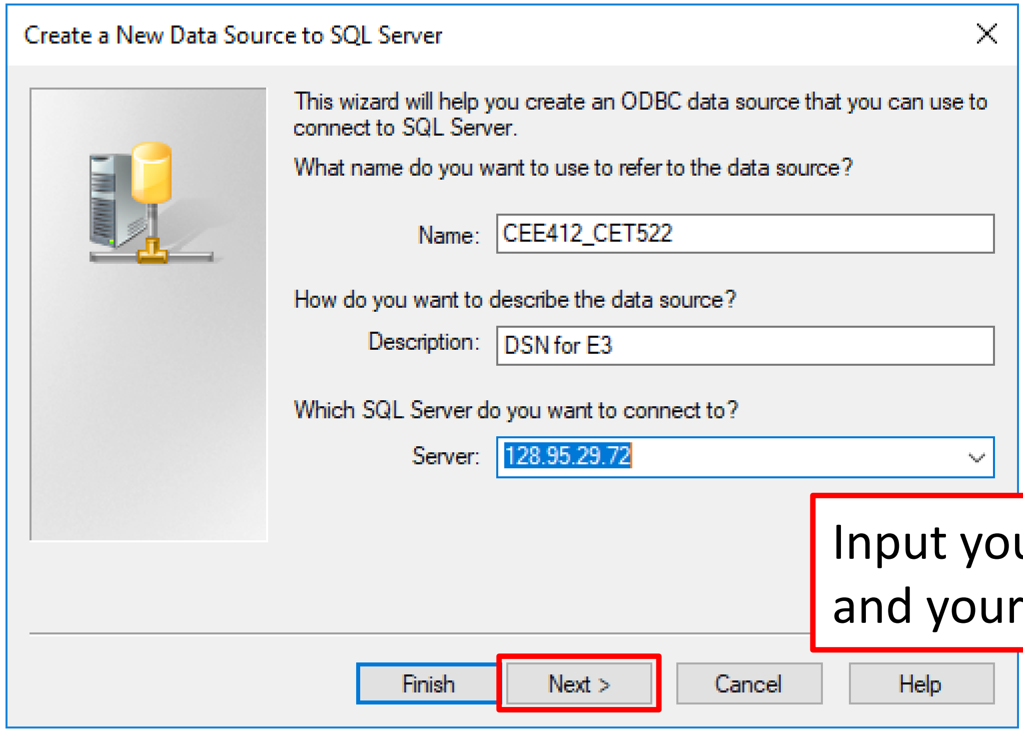

Create New Data Source

Select a driver for which you want to set up a data source.

Name	Vers
Microsoft Access dBASE Driver (*.dbf, *.ndx, *.mdx)	14.0
Microsoft Access Driver (*.mdb, *.accdb)	14.0
Microsoft Access Text Driver (*.txt, *.csv)	14.0
Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)	14.0
ODBC Driver 11 for SQL Server	2015
SQL Server	10.0
SQL Server Native Client 11.0	2011

Step1: Create a DSN

- Name your DSN something you will remember, use the class database IP, then click **Next**.
- Select SQL Server authentication and enter your SQL Server Login information, then click **Next**.



Create a New Data Source to SQL Server

This wizard will help you create an ODBC data source that you can use to connect to SQL Server.

What name do you want to use to refer to the data source?

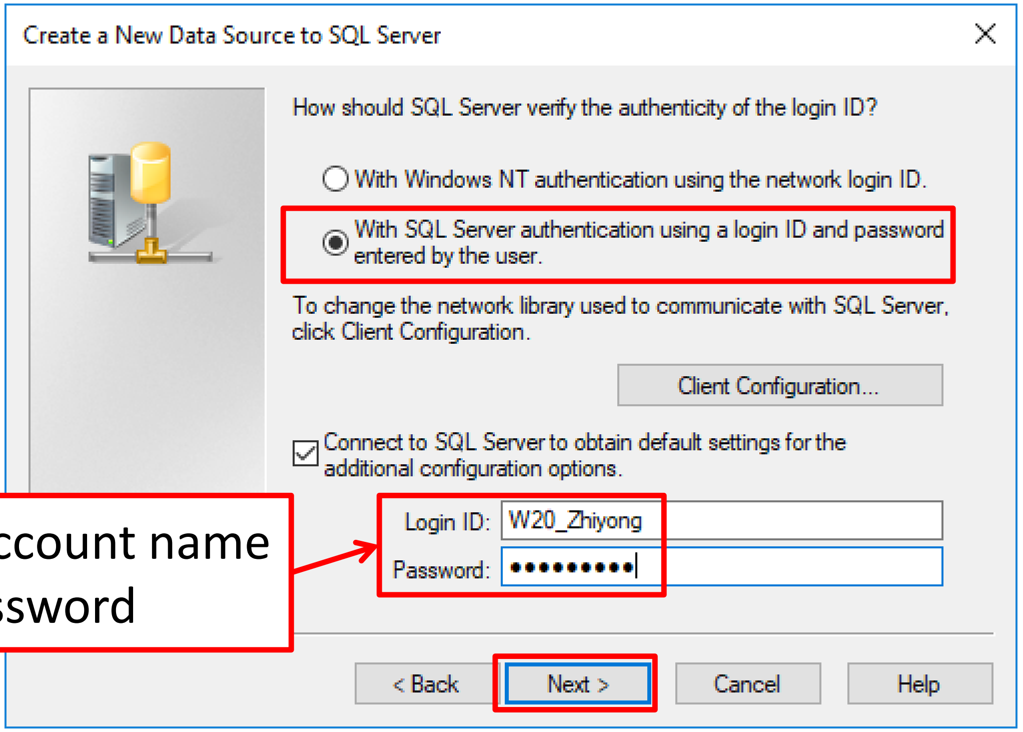
Name: CEE412_CET522

How do you want to describe the data source?

Description: DSN for E3

Which SQL Server do you want to connect to?

Server: 128.95.29.72



Create a New Data Source to SQL Server

How should SQL Server verify the authenticity of the login ID?

○ With Windows NT authentication using the network login ID.

● With SQL Server authentication using a login ID and password entered by the user.

To change the network library used to communicate with SQL Server, click Client Configuration.

☑ Connect to SQL Server to obtain default settings for the additional configuration options.

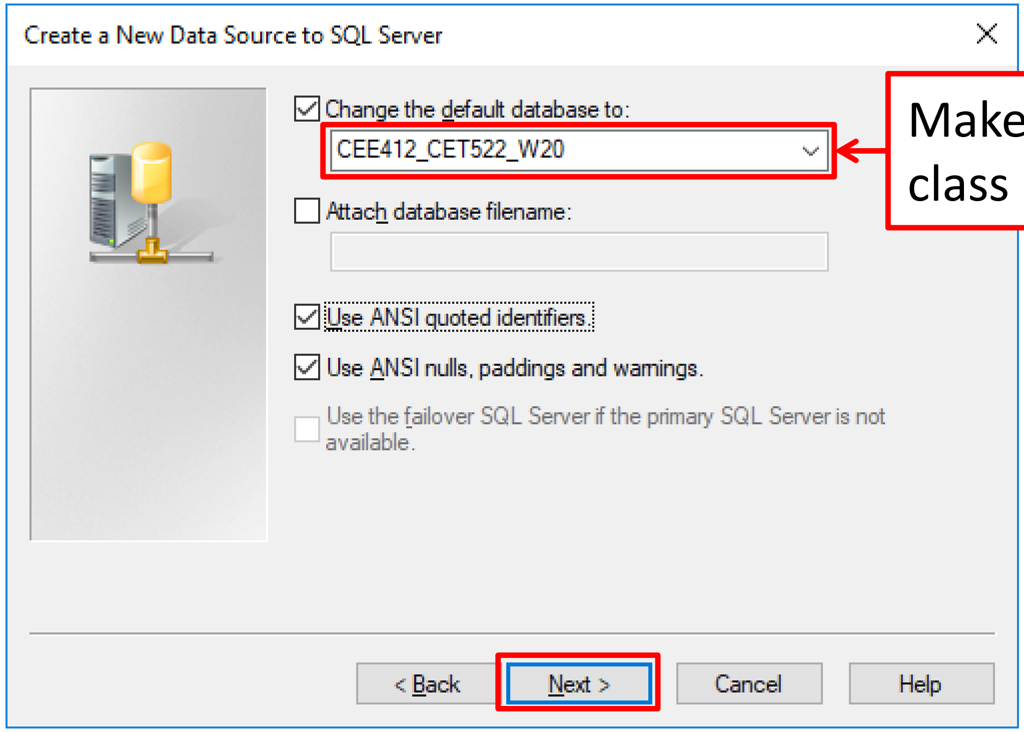
Login ID: W20_Zhiyong

Password: ●●●●●●●●●

Input your account name and your password

Step 1: Create a DSN

- Make sure the class database is the default database, leave everything else to default. Click **Next** and then **Finish**.



Create a New Data Source to SQL Server

☑ Change the default database to:
CEE412_CET522_W20

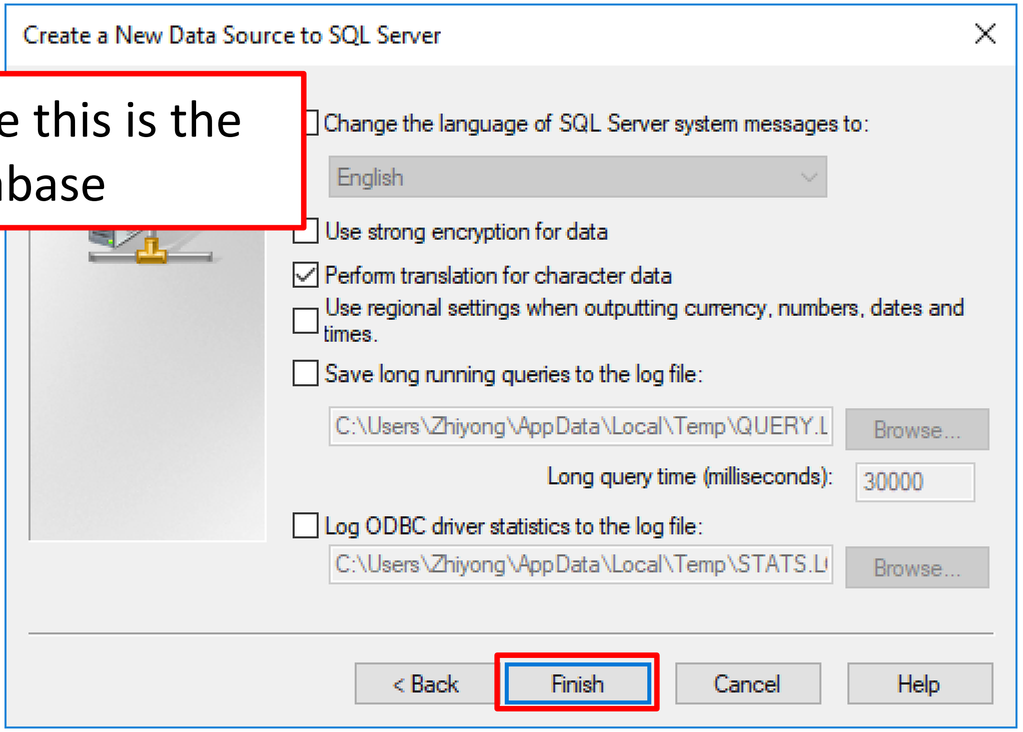
☐ Attach database filename:

☑ Use ANSI quoted identifiers.

☑ Use ANSI nulls, paddings and warnings.

☐ Use the failover SQL Server if the primary SQL Server is not available.

Make sure this is the class database



Create a New Data Source to SQL Server

☐ Change the language of SQL Server system messages to:
English

☐ Use strong encryption for data

☑ Perform translation for character data

☐ Use regional settings when outputting currency, numbers, dates and times.

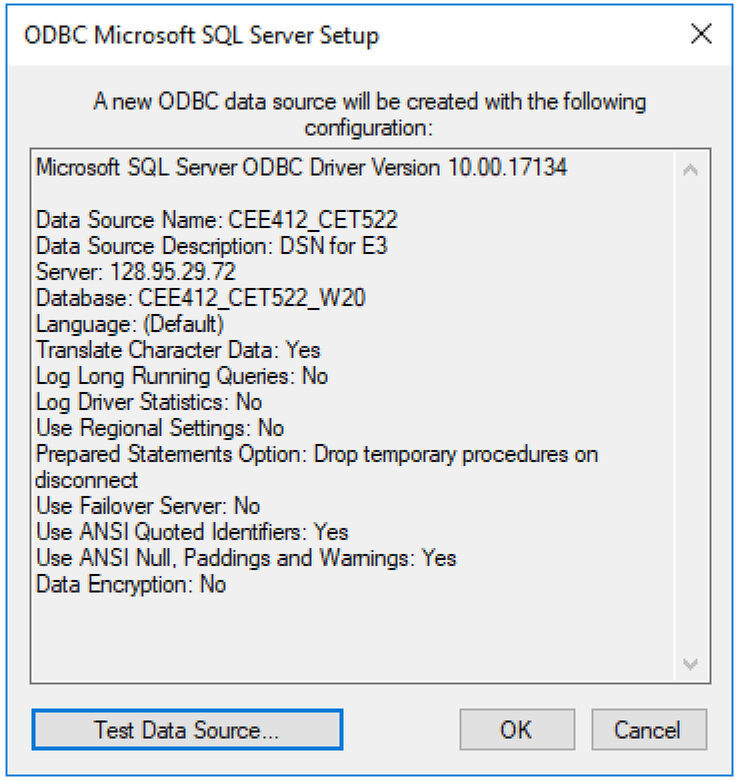
☐ Save long running queries to the log file:
C:\Users\Zhiyong\AppData\Local\Temp\QUERY.L

Long query time (milliseconds): 30000

☐ Log ODBC driver statistics to the log file:
C:\Users\Zhiyong\AppData\Local\Temp\STATS.L

Step 1: Create a DSN

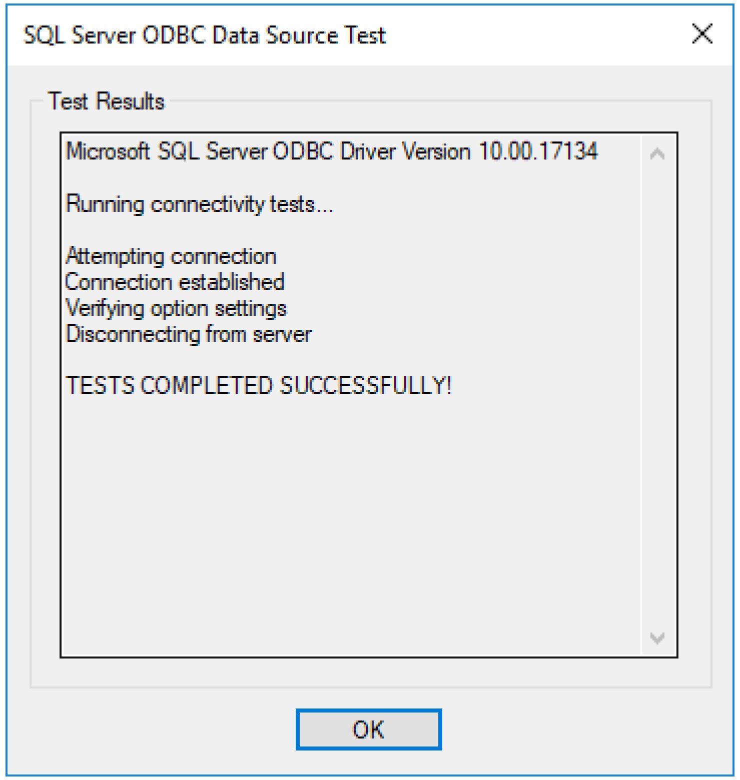
- You will see a summary of the connection, click "Test Data Source" to see if everything worked, then click **OK**.



A new ODBC data source will be created with the following configuration:

```
Microsoft SQL Server ODBC Driver Version 10.00.17134

Data Source Name: CEE412_CET522
Data Source Description: DSN for E3
Server: 128.95.29.72
Database: CEE412_CET522_W20
Language: (Default)
Translate Character Data: Yes
Log Long Running Queries: No
Log Driver Statistics: No
Use Regional Settings: No
Prepared Statements Option: Drop temporary procedures on
disconnect
Use Failover Server: No
Use ANSI Quoted Identifiers: Yes
Use ANSI Null, Paddings and Warnings: Yes
Data Encryption: No
```



```
Microsoft SQL Server ODBC Driver Version 10.00.17134

Running connectivity tests...

Attempting connection
Connection established
Verifying option settings
Disconnecting from server

TESTS COMPLETED SUCCESSFULLY!
```

Step 2: Create a Shiny App

- Create a Shiny App (R script) and import RODBC library

```r
library(shiny)
library(RODBC)

ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

- Enter the code shown below to create a connection object. You should change the function parameters based on your DSN and SQL Server login

```r
conn <- odbcConnect("CEE412_CET522", "Username", "Password")
```

DSN

SQL Server login

Step 3: Query Data in a Shiny App

- Query all data from the E4_Survey table in the CEE412_CET422_W20 database. Then, show the data entries in a table in the Shiny App
  1. First check the data types of all columns in the E4_Survey table
  2. Write a Query before the UI clause in your script

```
# query
surveyQuery <- "SELECT * FROM [E4_Survey]"

# select the Survey data based on query
surveyData <- sqlQuery(conn, surveyQuery)
```

  3. The data in the datetime type in SQL need to be converted into the right form. Write the following code to convert data:

```
# format the timestamp column into the format "%d/%m/%Y %H:%M:%S"
surveyData$timestamp <- as.character(as.POSIXct(surveyData$timestamp, origin="1970-01-01",
format="%d/%m/%Y %H:%M:%S"))
```

Check the two functions and try to understand what we are doing here

Step 3: Query Data in a Shiny App

- Adjust your UI and server function in your Shiny App to show the Query Result

```
# UI
ui <- fluidPage(

  # App title ----
  titlePanel("Hello Shiny!"),

  fluidRow(
    column(12,
      wellPanel(
        tableOutput('table')
      )
    )
  )
)
```

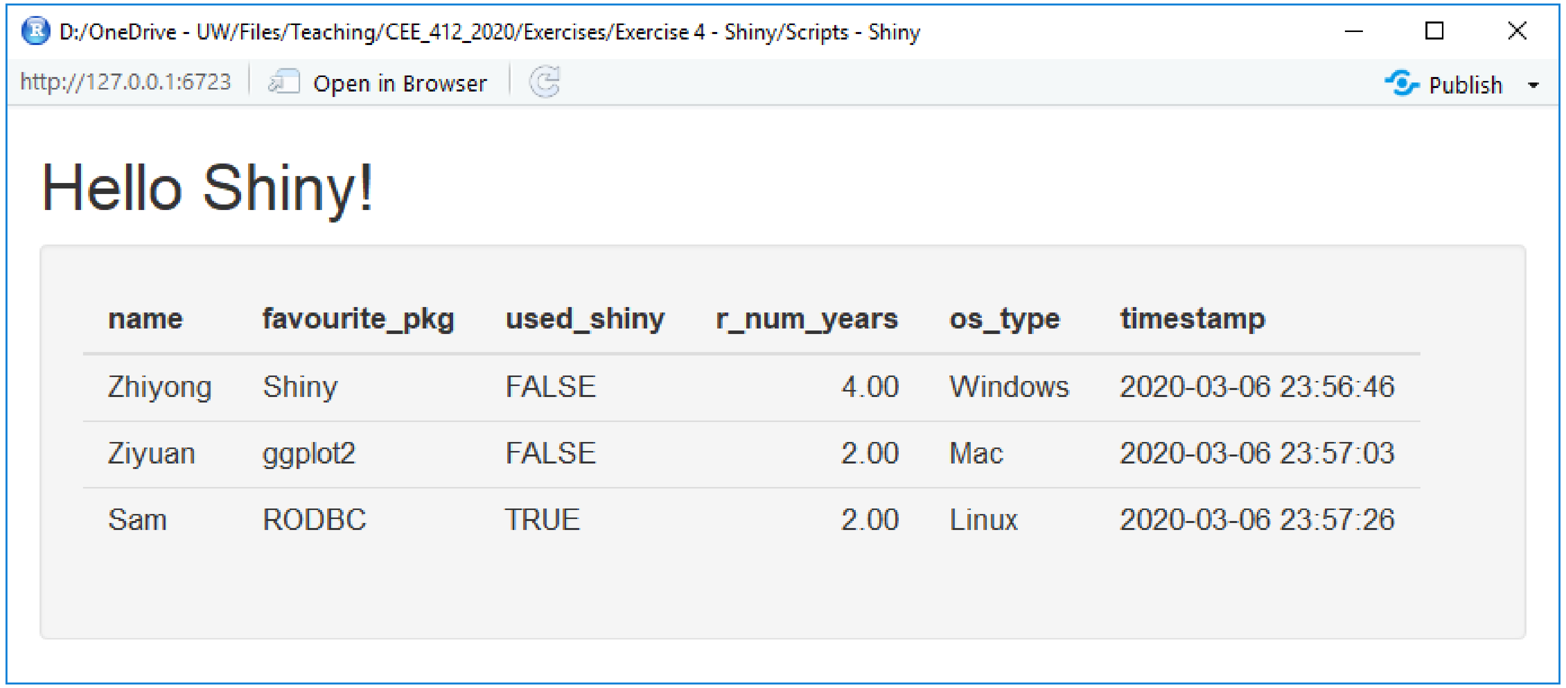
```
# Server
server <- function(input, output) {

  output$table <- renderTable(surveyData)

}
```

Step 3: Query Data in a Shiny App

- Run your code and check whether you can get a UI like the following figure:


D:/OneDrive - UW/Files/Teaching/CEE_412_2020/Exercises/Exercise 4 - Shiny/Scripts - Shiny
http://127.0.0.1:6723
Open in Browser
Publish

Hello Shiny!

name	favourite_pkg	used_shiny	r_num_years	os_type	timestamp
Zhiyong	Shiny	FALSE	4.00	Windows	2020-03-06 23:56:46
Ziyuan	ggplot2	FALSE	2.00	Mac	2020-03-06 23:57:03
Sam	RODBC	TRUE	2.00	Linux	2020-03-06 23:57:26

- If not, Check demo code file:
  - Exercises → Exercise 4 → Scripts → part_4_demo_1_QueryData.R

Step 4: Save Data

- If we want to do a survey, we can offer an online interface to the users to collect survey data in a more efficient way. In this step, we will create an interface to collect data by using Shiny and save the data into our database.

- Please copy/import the E4_Survey table into your own database. If you forget how to do that, please check Exercise II Part 2.
  - Do forget to create a **new DSN** to be used to connect to **your own database**.

- Now let's create a UI to let users input their survey results.

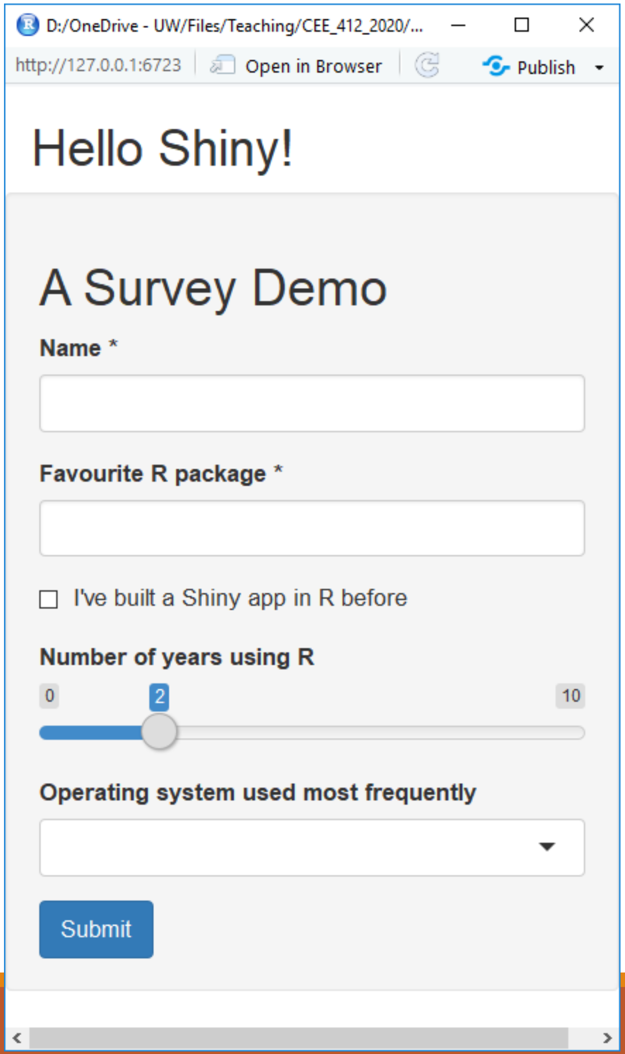
Step 4: Save Data

- As you checked, the E4_Survey table contains six columns, including
  - name: survey name
  - favourite_pkg: favourite R package
  - used_shiny: have the experience of using Shiny or not
  - r_num_years: number of years using R
  - os_type: Operating system used most frequently
  - timestamp: timestamp of the survey

- We want to design a proper UI to collect all the above six attributes.

Step 4: Save Data

- We want a UI look like this:



- UI components contain textInput, checkboxInput, sliderInput, selectInput, and actionButton

labelMandatory is a function to change the format of the label of the textInput

```
labelMandatory <- function(label) {
  tagList(
    label,
    span("*", class = "mandatory_star")
  )
}
```

```
fluidRow(
    column(6,
      wellPanel(
        titlePanel("A Survey Demo"),
        div(id = "form",
          textInput("name", labelMandatory("Name"), ""),
          textInput("favourite_pkg", labelMandatory("Favourite R package")),
          checkboxInput("used_shiny", "I've built a Shiny app in R before", FALSE),
          sliderInput("r_num_years", "Number of years using R", 0, 10, 2, ticks = FALSE),
          selectInput("os_type", "Operating system used most frequently",
                  c("",  "Windows", "Mac", "Linux")),
          actionButton("submit", "Submit", class = "btn-primary")
        )
      )
    )
  )
```

Step 4: Save Data

- If a user click on the submit, the input data should be saved into your own database. Here is what we need to do:
  - We define the save data process in a function, named as *saveDataToDatabase*, outside the UI and the server function. But this function will be called in the server function.

```
# save data to database
saveDataToDatabase <- function(data) {

  ColumnsOfTable      <- sqlColumns(conn, "E4_Survey")
  varTypes            <- as.character(ColumnsOfTable$TYPE_NAME)
  names(varTypes)     <- as.character(ColumnsOfTable$COLUMN_NAME)
  colnames(data)      <- as.character(ColumnsOfTable$COLUMN_NAME)

  sqlSave(conn, data, "E4_Survey", fast=TRUE, append=TRUE,  rownames=FALSE,
varTypes=varTypes )
}
```

"data" is the collected info that has been converted into a dataFrame. We will introduce it in the next slide

Get column names of E4_Survey

Get variable types (*varTypes*) and change the column name of "data" by using *colnames(data)*

Save data into your own E4_Survey table. (you can name your table whatever as you want)

Step 4: Save Data

- In the server function, we need to get the user's type-in or selection information via the ID of the input components, such as textInputs defined in the UI.

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

  # define the header
  fieldsAll <- c("name", "favourite_pkg", "used_shiny", "r_num_years", "os_type")

  # get the system time
  timestamp <- function() format(Sys.time(), "%Y-%m-%d %H:%M:%OS")

  # gather form data into a format
  formData <- reactive({
    data <- sapply(fieldsAll, function(x) input[[x]])
    data <- c(data, timestamp = timestamp())
    data <- t(data)
    data <- as.data.frame(data)
  })
```

Get a list of the names/IDs of the UI components, which will be used to extract the user's survey information from the **input**

Get the current timestamp in the format of "%Y-%m-%d %H:%M:%OS"

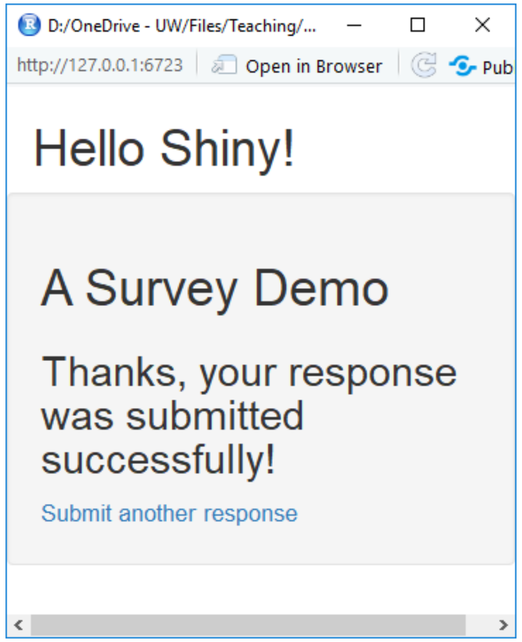
Extract the survey data from the input based on the fieldAll. Please chack the meaning of sapply function

Combine time stamp into the data

Transpose data. Please check t() function if needed.

Step 4: Save Data

- After finishing saving data, we want the UI give users feedback after they submit surveys, like the right figure
- We may need to use JavaScript to dynamically control the interface of the UI components.
- Shiny provide a package, **shinyjs**, to perform common useful JavaScript operations in Shiny apps. It can



Hello Shiny!

A Survey Demo

Thanks, your response was submitted successfully!

Submit another response

Hide (or show) an element	Disable (or enable) an input	Reset an input back to its original
Delay code execution by a few seconds	Easily call your own JavaScript functions from R	Many useful functions for both app developers and users

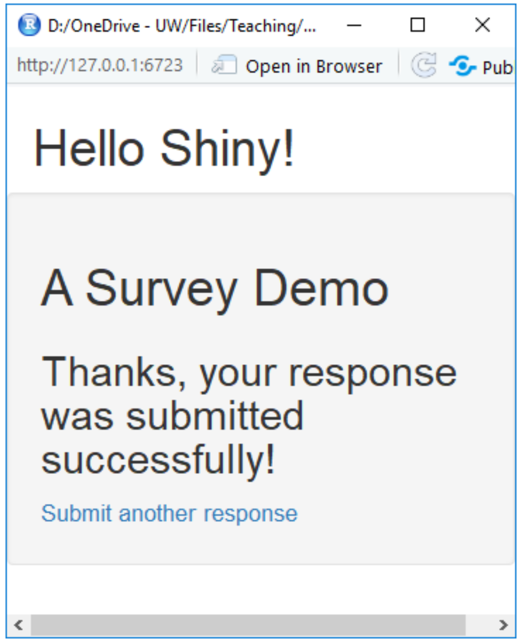
- Please check out more info and examples at: https://deanattali.com/shinyjs/

Step 4: Save Data

- We create a hidden panel/div to place the feedbacks.
  - div is a tag in HTML document to define a division or a section

```
fluidRow(
     column(6,
       wellPanel(
         titlePanel("A Survey Demo"),
         div(id = "form",
             …
         ) ,
         shinyjs::hidden(
           div(
             id = "thankyou_msg",
             h3("Thanks, your response was submitted successfully!"),
             actionLink("submit_another", "Submit another response")
           )
         )
       )
     )
   )
```

The hidden section will not be displayed unless the shinyjs settings are changed, which can be done in the server function.



Hello Shiny!

A Survey Demo

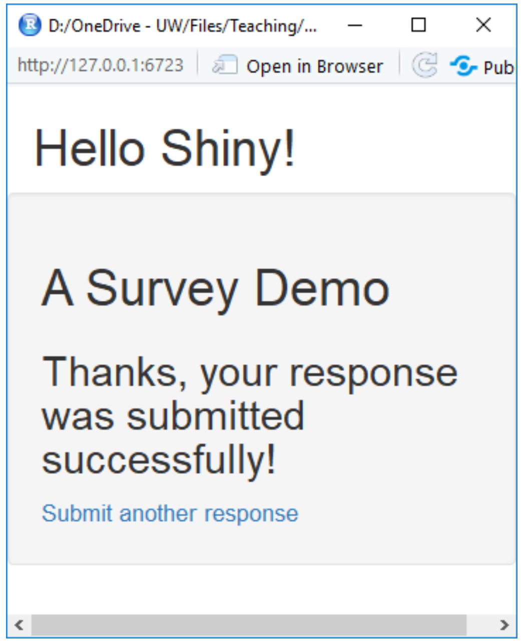
Thanks, your response was submitted successfully!

Submit another response

Step 4: Save Data

- Let’s see how to change the shinyjs settings in the server function:

```
server <- function(input, output) {
  …
  # action to take when submit button is pressed
  observeEvent(input$submit, {
    saveDataToDatabase(formData())
    shinyjs::reset("form")
    shinyjs::hide("form")
    shinyjs::show("thankyou_msg")
  })
  # action to take when submit_another button is pressed
  observeEvent(input$submit_another, {
    shinyjs::show("form")
    shinyjs::hide("thankyou_msg")
  })
}
```


D:/OneDrive - UW/Files/Teaching/...
http://127.0.0.1:6723

Hello Shiny!

A Survey Demo

Thanks, your response was submitted successfully!

Submit another response

Now, please combine the provided code into a Shiny App and try to run it.
If you encounter problems, try to find what might be the reason firstly . Then check & run the demo code:
Exercises → Exercise 4 → Scripts → part_4_demo_2_RODBC.R

Step 4: Save Data

- Now, please combine the provided code into a Shiny App and try to run it.

- If you encounter problems, try to find what might be the reasons at first. Then, you can check and run the demo code:
  - Exercises → Exercise 4 → Scripts → part_4_demo_2_RODBC.R

Step 5: Display your submission

- If you are the administrator of the Shiny Survey App, you want to see the most recent three survey results on the Shiny App. The Shiny App also provides you an interface to input your survey.

- After submitting a new survey via the App, you want to see the updated most recent three survey results by just clicking on a button.

- How can you develop such an interface/App for yourself?
  - Please try to combine the tools/demos you have tried in Exercise 4 Part 2.
  - Before moving to next page, please try your best to develop such an App.

Step 5: Display your submission

- Please check the demo code file:
  - Exercises → Exercise 4 → Scripts → part_4_demo_3_VisualizeData.R
- In the demo code, please compare the output$table1 and output$table2 in the server function and think about the main difference between them
- The updatedData in the server function is an eventReactive function, the last row in the {}, i.e. queryData, is the output/returned data, which are represented by updatedData().

```
 updatedData <- eventReactive(input$refresh, {
   queryData <- sqlQuery(conn, surveyQuery)
   queryData$timestamp <-
as.character(as.POSIXct(queryData$timestamp, origin="1970-01-
01", format="%d/%m/%Y %H:%M:%S"))
   queryData
  })
```

Other Database Connection Methods

- Shiny also supports other database connection methods/packages, including odbc, RODBC, DBI, and etc.
- We will introduce you another way to connect to database, i.e. the odbc package.
  - Please check out the demo code file:
  - Exercises → Exercise 4 → Scripts → part_4_demo_4_ODBC.R
    - This demo has the same functionality with the part_4_demo_2_RODBC.R. We would like to show you the difference between these two packages.
    - By using odbc, you do not need to build DSN first. It is more convenience for you.

  - I've mentioned the R Shiny Contest (https://blog.rstudio.com/2020/02/12/shiny-contest-2020-is-here/), which is highly related to our project. If you want to participate it, you need to deploy your app on shinyapps.io (https://www.shinyapps.io/) and you may need to use odbc.

Notes

- Using a Windows computer would be easier for you to connect to the database in R/Shiny. Thus, we recommend you to use a Windows computer to finish this exercise or even your final project.
- If you have to use a Mac, there are solutions. But it would be more complicate and there would be more bugs or incompatible problems.
- If needed, I can post a tutorial about how to use R to connect our database server on your Mac.
- Next step, another exercise on visualizing data on maps and etc.