R Connection with MySQL

.

Working with R and MySQL

```
# Install packages
# install.packages("RMySQL")
# install.packages("RODBC")
# install.packages("DBI")
# install.packages("odbc")

# Load package
library(RMySQL)
```

```
## Warning: package 'RMySQL' was built under R version 4.3.2
```

```
## Loading required package: DBI
```

```
## Warning: package 'DBI' was built under R version 4.3.2
```

```
library(RODBC)
```

```
## Warning: package 'RODBC' was built under R version 4.3.2
```

```
library(DBI)
library(odbc)
```

```
## Warning: package 'odbc' was built under R version 4.3.2
```

Connect to database

Connect to the database from R

```
# Connect to database
mydb <- dbConnect(MySQL(), user = 'root',
                  password = 'yours',
                  dbname = 'weather_db',
                  host = 'localhost')
```

Table manipulation

We will use "dbListTables" function to list all tables in our database and "dbWriteTable" to write a table from a data frame

```
dbListTables(mydb)

# dbWriteTable(mydb, "mtcars",
#              mtcars[1:5, ])
```

Error?

Once the above problem happens, you can try the following code to resolve it:

```
# dbSendQuery(mydb, "SET GLOBAL local_infile = true;")
```

Then, remove the "mtcars" from the weather_db using:

```
# dbRemoveTable(mydb, "mtcars")
```

```
dbWriteTable(mydb, "mtcars",mtcars[1:5, ])
```

```
## [1] TRUE
```

We will use "dbExistTable" function to check whether a table exists and "dbRemoveTable" to drop a table

```
dbExistsTable(mydb, "mtcars")
```

```
## [1] TRUE
```

```
# dbRemoveTable(mydb, "mtcars")
```

Exploring data within a table

We will use dbListFields function to list fields within a table and dbSendQuery to send a query to the data base

```
dbListFields(mydb,
             'metoffice_dailyweatherdata')
```

```
##  [1] "LocationId"       "obs_dateTime"     "obs_date"         "obs_time"
##  [5] "temperature"      "windspeed"        "humidity"         "dewpoint"
##  [9] "pressure"         "windgust"         "visibility"       "winddirection"
## [13] "pressuretendency" "timestamp"        "rainy"            "windy"
## [17] "snow"             "weatherType"
```

```
# dbSendQuery(mydb, 'Select * from country')
```

Fetch Results to R

We will use "dbFetch" function to fetch results from a query to a data frame in R

```
Data_frame = dbFetch(dbSendQuery(Connection_name,
                            "SQL query"),n=number_of_rows)
```

```
# country_table = dbFetch(dbSendQuery(mydb,
#                                     'Select * from country'),n=-1)
```

n = -1 or n = Inf is to retrieve all pending records.

Error?

```
# dbClearResult(dbListResults(mydb)[[1]])
```

TEST with dbFetch:

```
country_table = dbFetch(dbSendQuery(mydb,
                                    'Select * from country'),n=-1)
```

```
loc_temp = dbFetch(dbSendQuery(mydb,
                                  'SELECT c.location, m.obs_date,
                                          m.temperature
                                   FROM cat_locations AS c
                                   INNER JOIN metoffice_dailyweatherdata AS m
                                   ON c.LocationID = m.LocationId LIMIT 10'),n=-1)
```

SESSION 6

Working with R and SQL

We will use "dbListTables" function to list all tables in our database and "dbWriteTable" to write a table from a data frame

```
dbListTables(mydb)
```

```
##  [1] "carlisle_max_temperature"     "cat_locations"
##  [3] "cat_postcode_latlong"         "cat_regions"
##  [5] "country"                      "high_pressure_percentages"
##  [7] "metoffice_dailyweatherdata"   "metoffice_forecast_text"
##  [9] "mtcars"                       "postcodelatlng"
## [11] "rain_or_snow_weather_station" "rainy_snowy"
## [13] "snow_weather_station_counts"  "tempw"
## [15] "timezone"                     "weathertype"
## [17] "zones"
```

dbListFields(Connection_name,Table_name) Lists the attributes of a table.

dbReadTable reads a table as a data.frame.

```
dbListFields(mydb, "metoffice_dailyweatherdata")
```

```
##  [1] "LocationId"       "obs_dateTime"     "obs_date"         "obs_time"
##  [5] "temperature"      "windspeed"        "humidity"         "dewpoint"
##  [9] "pressure"         "windgust"         "visibility"       "winddirection"
## [13] "pressuretendency" "timestamp"        "rainy"            "windy"
## [17] "snow"             "weatherType"
```

```
dftempw <- dbReadTable(mydb, "tempw")
print(class(dftempw)) # we extract the tempw table as dataframe
```

```
## [1] "data.frame"
```

```
summary(dftempw) # numerical summaries for each attribute
```

```
##    LocationId   obs_dateTime         obs_date           obs_time
##  Min.   :3002   Length:16          Length:16          Length:16
##  1st Qu.:3504   Class :character   Class :character   Class :character
##  Median :3672   Mode  :character   Mode  :character   Mode  :character
##  Mean   :3504
##  3rd Qu.:3672
##  Max.   :3672
##
##   temperature      windspeed        humidity        dewpoint
##  Min.   :4.500   Min.   : 2.00   Min.   :71.70   Min.   :0.00
##  1st Qu.:5.450   1st Qu.: 4.50   1st Qu.:87.38   1st Qu.:0.00
##  Median :6.400   Median : 6.00   Median :91.80   Median :0.00
##  Mean   :6.231   Mean   :10.31   Mean   :88.82   Mean   :1.00
##  3rd Qu.:6.725   3rd Qu.:10.75   3rd Qu.:93.95   3rd Qu.:0.75
##  Max.   :8.300   Max.   :31.00   Max.   :95.50   Max.   :5.00
##
##     pressure         windgust        visibility    winddirection
##  Min.   :   0.0   Min.   : 0.000   Min.   :    0   Length:16
##  1st Qu.:   0.0   1st Qu.: 0.000   1st Qu.:    0   Class :character
##  Median :   0.0   Median : 0.000   Median :    0   Mode  :character
##  Mean   : 251.4   Mean   : 9.438   Mean   : 4000
##  3rd Qu.: 251.2   3rd Qu.: 8.000   3rd Qu.: 2750
##  Max.   :1007.0   Max.   :41.000   Max.   :27000
##
##  pressuretendency     rainfall          radiation  timestamp
##  Length:16          Length:16          Min.   :0   Length:16
##  Class :character   Class :character   1st Qu.:0   Class :character
##  Mode  :character   Mode  :character   Median :0   Mode  :character
##                                        Mean   :0
##                                        3rd Qu.:0
##                                        Max.   :0
##                                        NA's   :4
```

Copying R Data frames to a Database

We can use the dbWriteTable function to create a database table from R (data frame). We will use one of the tables (mtcars) that exist in R

- dbWriteTable(Connection_name,"table_name", data_frame)

```
data(mtcars)
head(mtcars) # print the first few rows
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
# dbWriteTable(dbcon,"mtcars",mtcars) # writes the dataframe to the database
dfcars <- dbReadTable(mydb,"mtcars") # read the table from the database
summary(dfcars) # produce numerical summaries
```

```
##       mpg             cyl         disp             hp             drat
##  Min.   :18.70   Min.   :4   Min.   :108.0   Min.   : 93.0   Min.   :3.080
##  1st Qu.:21.00   1st Qu.:6   1st Qu.:160.0   1st Qu.:110.0   1st Qu.:3.150
##  Median :21.00   Median :6   Median :160.0   Median :110.0   Median :3.850
##  Mean   :20.98   Mean   :6   Mean   :209.2   Mean   :119.6   Mean   :3.576
##  3rd Qu.:21.40   3rd Qu.:6   3rd Qu.:258.0   3rd Qu.:110.0   3rd Qu.:3.900
##  Max.   :22.80   Max.   :8   Max.   :360.0   Max.   :175.0   Max.   :3.900
##        wt             qsec             vs              am          gear
##  Min.   :2.320   Min.   :16.46   Min.   :0.0   Min.   :0.0   Min.   :3.0
##  1st Qu.:2.620   1st Qu.:17.02   1st Qu.:0.0   1st Qu.:0.0   1st Qu.:3.0
##  Median :2.875   Median :17.02   Median :0.0   Median :1.0   Median :4.0
##  Mean   :2.894   Mean   :17.71   Mean   :0.4   Mean   :0.6   Mean   :3.6
##  3rd Qu.:3.215   3rd Qu.:18.61   3rd Qu.:1.0   3rd Qu.:1.0   3rd Qu.:4.0
##  Max.   :3.440   Max.   :19.44   Max.   :1.0   Max.   :1.0   Max.   :4.0
##       carb
##  Min.   :1.0
##  1st Qu.:1.0
##  Median :2.0
##  Mean   :2.4
##  3rd Qu.:4.0
##  Max.   :4.0
```

Removing a table

- We can check if a table exists in the database with dbExistsTable(Connection_name,"table_name")
- we can remove a table using the dbRemoveTable(connection_name, "table_name") function:

```
# if the table exists remove it
if(dbExistsTable(mydb,"mtcars")){
  dbRemoveTable(mydb, "mtcars")
}else{
  print("Table does not exist")
}
```

```
## [1] TRUE
```

Running SQL queries

We can run SQL queries with dbSendQuery and with dbFetch we can fetch all the results. Finally we can disconnect the database with dbDisconnect.

```
res<- dbSendQuery(mydb,"SELECT * FROM metoffice_dailyweatherdata WHERE LocationId='3002'")

dfres<- dbFetch(res)

summary(dfres)
```

```
##    LocationId   obs_dateTime         obs_date           obs_time
##  Min.   :3002   Length:79          Length:79          Length:79
##  1st Qu.:3002   Class :character   Class :character   Class :character
##  Median :3002   Mode  :character   Mode  :character   Mode  :character
##  Mean   :3002
##  3rd Qu.:3002
##  Max.   :3002
##   temperature      windspeed        humidity        dewpoint
##  Min.   :1.400   Min.   :18.00   Min.   :49.20   Min.   :-6.200
##  1st Qu.:3.200   1st Qu.:24.00   1st Qu.:73.95   1st Qu.:-0.650
##  Median :7.500   Median :28.00   Median :83.30   Median : 4.900
##  Mean   :6.254   Mean   :28.71   Mean   :79.06   Mean   : 2.816
##  3rd Qu.:8.900   3rd Qu.:32.00   3rd Qu.:86.25   3rd Qu.: 6.600
##  Max.   :9.800   Max.   :45.00   Max.   :94.60   Max.   : 7.500
##     pressure       windgust       visibility    winddirection
##  Min.   : 986   Min.   :29.00   Min.   : 2600   Length:79
##  1st Qu.: 994   1st Qu.:37.00   1st Qu.:11000   Class :character
##  Median :1001   Median :41.00   Median :13000   Mode  :character
##  Mean   :1002   Mean   :43.92   Mean   :14615
##  3rd Qu.:1010   3rd Qu.:47.50   3rd Qu.:18000
##  Max.   :1018   Max.   :66.00   Max.   :30000
##  pressuretendency    timestamp           rainy              windy
##  Length:79          Length:79          Length:79          Length:79
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##      snow           weatherType
##  Length:79          Length:79
##  Class :character   Class :character
##  Mode  :character   Mode  :character
```

```
##
##
##
```

Disconnect database

```
# dbDisconnect(mydb)
```

Databases with tidyverse

We can also use a database conection with tidyverse (e.g. %>% pipes) using the library dplyr:

```
library(dplyr)
```

We can then create a reference tible to one of the databases tables:

```
tweather <- tbl(mydb,"metoffice_dailyweatherdata")
tweather
```

```
## # Source:   table<metoffice_dailyweatherdata> [?? x 18]
## # Database: mysql 8.0.33 [@localhost:/weather_db]
##    LocationId obs_dateTime      obs_date obs_time temperature windspeed humidity
##         <int> <chr>             <chr>    <chr>          <dbl>     <int>    <dbl>
##  1       3002 2020-01-01 00:00~ 2020-01~ 0:00:00          7.5        21     84
##  2       3002 2020-01-01 01:00~ 2020-01~ 1:00:00          7.5        22     81.7
##  3       3002 2020-01-01 02:00~ 2020-01~ 2:00:00          7.9        24     79.9
##  4       3002 2020-01-01 03:00~ 2020-01~ 3:00:00          7.5        23     82.3
##  5       3002 2020-01-01 04:00~ 2020-01~ 4:00:00          8          18     84.6
##  6       3002 2020-01-01 05:00~ 2020-01~ 5:00:00          8.3        24     85.3
##  7       3002 2020-01-01 06:00~ 2020-01~ 6:00:00          8.5        30     89
##  8       3002 2020-01-01 07:00~ 2020-01~ 7:00:00          8.7        28     89.6
##  9       3002 2020-01-01 08:00~ 2020-01~ 8:00:00          8.7        23     88.4
## 10       3002 2020-01-01 09:00~ 2020-01~ 9:00:00          9.1        29     84.3
## # i more rows
## # i 11 more variables: dewpoint <dbl>, pressure <int>, windgust <int>,
## #   visibility <int>, winddirection <chr>, pressuretendency <chr>,
## #   timestamp <chr>, rainy <chr>, windy <chr>, snow <chr>, weatherType <chr>
```

Select according to locationid and group by date to display average temperature and windspeed

```
tweather %>%
    filter(LocationId==3002) %>% # selects only the records that contain Location id 3002
    select(obs_date,temperature,windspeed) %>% # selects 3 attributes
    group_by(obs_date) %>%  # group by diferent dates
    summarize(AverageTemp = mean(temperature),
              Averagewind = mean(windspeed)) # for each group print the means
```

```
## # Source:   SQL [4 x 3]
## # Database: mysql 8.0.33 [@localhost:/weather_db]
##   obs_date   AverageTemp Averagewind
##   <chr>            <dbl>       <dbl>
```

```
## 1 2020-01-01        8.57        25.8
## 2 2020-01-02        8.03        27.5
## 3 2020-01-03        3.22        34.1
## 4 2020-01-04        2.64        24.3
```

R TASKS

TASK 1

Using R to work on the database "weather.db". Based on the tables "timezones" and "zones", please find out the distinct names of all time zones which have within +-2 hours difference from London (GMT).

STEP A: Extract the two tables from database as data frames in R. Then, based on these two data frames, finding the required time zones using "joins" from R package "dplyr".

STEP B: Using function "dbFetch" in R to manipulate the data in MySQL, and return the resulting table as data frame in R.

Remark 1: that column "gmt_offset" in table "timezone" gives the information for time zones. In "gmt_offset", 0 means the London (GMT), 3600 is "3600 seconds" (1 hour), and 7200 is 2×3600 seconds" (2 hours).

Remark 2: You can use R package "dplyr" to get the function called "inner_join()"

```
#Extract tables for the Data Base
timezones = dbFetch(dbSendQuery(mydb, 'SELECT * FROM timezone'),n=-1)
head(timezones)
```

STEP A

```
##   zone_id abbreviation  time_start gmt_offset dst
## 1       1          WET          NA          0   0
## 2       1          WET -2147397248          0   0
## 3       1          CET  -733881600       3600   0
## 4       1         CEST   481078800       7200   1
## 5       1          CET   496803600       3600   0
## 6       1         CEST   512528400       7200   1
```

```
zones = dbFetch(dbSendQuery(mydb, 'SELECT * FROM zones'),n=-1)

head(zones)
```

```
##   zone_id country_code        zone_name
## 1       1           AD   Europe/Andorra
## 2       2           AE       Asia/Dubai
## 3       3           AF       Asia/Kabul
## 4       4           AG  America/Antigua
## 5       5           AI America/Anguilla
## 6       6           AL    Europe/Tirane
```

```r
#Filter timezones that are +-2 hours apart from London time
index.1 <- which(((timezones$gmt_offset>=-2*3600)&(timezones$gmt_offset <= 2* 3600))==1)

timezones_near_london <-timezones[index.1, ]

tail(timezones_near_london) # tail prints last 5 observations
```

```
##      zone_id abbreviation time_start gmt_offset dst
## 9989     142          EET  591667200       7200   0
## 9991     142          EET  623203200       7200   0
## 9993     142          EET  654739200       7200   0
## 9995     142          EET  686275200       7200   0
## 9997     142          EET  717897600       7200   0
## 9999     142          EET  749433600       7200   0
```

```r
# Change the row names (to avoid jumps, e.g. from row 338 to 512)
row.names(timezones_near_london) <- as.character(1:nrow(timezones_near_london))
```

```r
#Join with zones data frame to get the names of the zones
zones_near_london_full <- inner_join(timezones_near_london, zones,by = "zone_id")

#Select distinct names of those zones
zones_near_london <- distinct(zones_near_london_full, zone_name)
zones_near_london
```

```
##                         zone_name
## 1                  Europe/Andorra
## 2                   Europe/Tirane
## 3                   Africa/Luanda
## 4                Antarctica/Palmer
## 5                 Antarctica/Troll
## 6  America/Argentina/Buenos_Aires
## 7        America/Argentina/Cordoba
## 8          America/Argentina/Salta
## 9          America/Argentina/Jujuy
## 10       America/Argentina/Tucuman
## 11     America/Argentina/Catamarca
## 12      America/Argentina/La_Rioja
## 13      America/Argentina/San_Juan
## 14       America/Argentina/Mendoza
## 15      America/Argentina/San_Luis
## 16 America/Argentina/Rio_Gallegos
## 17       America/Argentina/Ushuaia
## 18                   Europe/Vienna
## 19                Europe/Mariehamn
## 20                 Europe/Sarajevo
## 21                 Europe/Brussels
## 22              Africa/Ouagadougou
## 23                   Europe/Sofia
## 24                Africa/Bujumbura
## 25               Africa/Porto-Novo
## 26                 America/Noronha
```

```
## 27                 America/Belem
## 28              America/Fortaleza
## 29                 America/Recife
## 30              America/Araguaina
## 31                 America/Maceio
## 32                  America/Bahia
## 33              America/Sao_Paulo
## 34                Africa/Gaborone
## 35                   Europe/Minsk
## 36               America/St_Johns
## 37              America/Goose_Bay
## 38            America/Pangnirtung
## 39                Africa/Kinshasa
## 40              Africa/Lubumbashi
## 41                  Africa/Bangui
## 42             Africa/Brazzaville
## 43                  Europe/Zurich
## 44                 Africa/Abidjan
## 45                  Africa/Douala
## 46            Atlantic/Cape_Verde
## 47                   Asia/Nicosia
## 48                  Europe/Prague
## 49                  Europe/Berlin
## 50                Europe/Busingen
## 51              Europe/Copenhagen
## 52                 Africa/Algiers
## 53                 Europe/Tallinn
## 54                   Africa/Cairo
```

```
sql_zones_near_london = dbFetch(dbSendQuery(mydb,
'SELECT DISTINCT zone_name
FROM
    (SELECT
        *
    FROM
        timezone
    WHERE
        gmt_offset >= - 2*3600
            AND gmt_offset <= 2*3600) l_time
    INNER JOIN zones zon
    ON zon.zone_id = l_time.zone_id'), n=-1)

sql_zones_near_london = sql_zones_near_london
```

STEP B

TASK 2

Write your results, one data frame obtained in question 1 from either task 1 or 2, as a table in weather.db

```
#Write the new table in weather_db
dbSendQuery(mydb, "SET GLOBAL local_infile = true;")
```

```
## <MySQLResult:12,0,21>
```

```
dbWriteTable(mydb, "sql_zones_near_london", sql_zones_near_london)
```

```
## [1] TRUE
```

TASK 3

Based on question 2, list all the tables and check whether your newly created table exists in the weather_db (Hint: using dbListTables and dbExistsTable).

```
#List tables
print(dbListTables(mydb))
```

```
##  [1] "carlisle_max_temperature"     "cat_locations"
##  [3] "cat_postcode_latlong"         "cat_regions"
##  [5] "country"                      "high_pressure_percentages"
##  [7] "metoffice_dailyweatherdata"   "metoffice_forecast_text"
##  [9] "postcodelatlng"               "rain_or_snow_weather_station"
## [11] "rainy_snowy"                  "snow_weather_station_counts"
## [13] "sql_zones_near_london"        "tempw"
## [15] "timezone"                     "weathertype"
## [17] "zones"
```

```
#Check if a table exists
dbExistsTable(mydb, "sql_zones_near_london")
```

```
## [1] TRUE
```

TASK 4

Remove the new table from weather_db (Hint use dbRemoveTable)

```
dbRemoveTable(mydb, "sql_zones_near_london")
```

```
## [1] TRUE
```

REFERENCE: MA332-_-SU : Databases and data processing with SQL (Summer), University of Essex, https://colab.research.google.com/drive/11BHiwNKEL7-mGoIYeRI9pula8vTjT1kk?usp=sharing, 2023.