PDU Praca Domowa nr 1

Kacper Staroń

15.04.2019

1 Wstęp

W poniższym dokumencie przedstawione zostanie moje rozwiązanie pracy domowej nr 1. z przedmiotu Przetwarzanie Danych Ustrukturyzowanych (rok akademicki 2018/19, semestr letni). Zawiera ono implementacje odpowiadające 7 zapytaniom SQLite przy użyciu funkcji bazowych języka R oraz pakietów data.table i dplyr, komentarze i analizę czasu działania.

2 Pobieranie danych

Rozwiązywane poniżej zadanie polega na filtrowaniu i analizie uproszczonych zestawów danych z serwisu https://travel.stackexchange.com/, zapisanych w postaci obiektów typu data.frame i data.table Oprócz samych analizowanych danych wymagane jest zaimportowanie odpowiednich pakietów.

```r
library(sqldf)
library(dplyr)
library(tidyverse)
library(data.table)

options(stringsAsFactors=FALSE)

Posts <- read.csv("C:/Users/staro/Desktop/PDU/pracadomowa1/Posts.csv.gz")
Comments <- read.csv("C:/Users/staro/Desktop/PDU/pracadomowa1/Comments.csv.gz")
Votes <- read.csv("C:/Users/staro/Desktop/PDU/pracadomowa1/Votes.csv.gz")
Users <- read.csv("C:/Users/staro/Desktop/PDU/pracadomowa1/Users.csv.gz")
Badges <- read.csv("C:/Users/staro/Desktop/PDU/pracadomowa1/Badges.csv.gz")

PostsDT <- setDT(read.csv("C:/Users/staro/Desktop/PDU/pracadomowa1/Posts.csv.gz"))
CommentsDT <- setDT(read.csv("C:/Users/staro/Desktop/PDU/pracadomowa1/Comments.csv.gz"))
VotesDT <- setDT(read.csv("C:/Users/staro/Desktop/PDU/pracadomowa1/Votes.csv.gz"))
UsersDT <- setDT(read.csv("C:/Users/staro/Desktop/PDU/pracadomowa1/Users.csv.gz"))
BadgesDT <- setDT(read.csv("C:/Users/staro/Desktop/PDU/pracadomowa1/Badges.csv.gz"))
```

3 Polecenia

1. Wybieramy informacje o autorach 10 najpopularniejszych (według "Favorite") postów.
2. Wybieramy tytuły i ID 10 postów o największej ilości odpowiedzi.

3. Wybieramy tytuły i ID postów o największej ilości upvote'ów z każdego roku.
4. Wybieramy posty o różnicy przynajmniej 50 między wynikiem (Score) pytania i najwyżej ocenionej odpowiedzi na nie.
5. Wybieramy 10 postów o największej sumie wyników (Score) komentarzy (Comments) do nich.
6. Wybieramy użytkowników posiadających między 2 a 10 odznak (Badges).
7. Wybieramy 10 postów, które mają najwięcej upvote'ów sprzed 2016 roku i żadnych później.

4 Implementacje

4.1 SQLite

Zadanie 1.

```
df_sql_1 <- function(df1, df2){
  Posts <- df1
  Users <- df2

  sqldf("SELECT
        Users.DisplayName,
        Users.Age,
        Users.Location,
        SUM(Posts.FavoriteCount) AS FavoriteTotal,
        Posts.Title AS MostFavoriteQuestion,
        MAX(Posts.FavoriteCount) AS MostFavoriteQuestionLikes
        FROM Posts
        JOIN Users ON Users.Id=Posts.OwnerUserId
        WHERE Posts.PostTypeId=1
        GROUP BY OwnerUserId
        ORDER BY FavoriteTotal DESC
        LIMIT 10") -> df
}
```

Zadanie 2.

```
df_sql_2 <- function(df1){

    Posts <- df1

    sqldf("SELECT
          Posts.ID,
          Posts.Title,
          Posts2.PositiveAnswerCount
          FROM Posts
          JOIN (
          SELECT
          Posts.ParentID,
          COUNT(*) AS PositiveAnswerCount
          FROM Posts
          WHERE Posts.PostTypeID=2 AND Posts.Score>0
          GROUP BY Posts.ParentID
          ) AS Posts2
          ON Posts.ID=Posts2.ParentID
          ORDER BY Posts2.PositiveAnswerCount DESC
          LIMIT 10") -> df
  }
```

Zadanie 3.

```
df_sql_3 <- function(df1, df2){
  Posts <- df1
  Votes <- df2

  sqldf("SELECT
        Posts.Title,
        UpVotesPerYear.Year,
        MAX(UpVotesPerYear.Count) AS Count
        FROM (
        SELECT
        PostId,
        COUNT(*) AS Count,
        STRFTIME('%Y', Votes.CreationDate) AS Year
        FROM Votes
        WHERE VoteTypeId=2
        GROUP BY PostId, Year
        ) AS UpVotesPerYear
        JOIN Posts ON Posts.Id=UpVotesPerYear.PostId
        WHERE Posts.PostTypeId=1
        GROUP BY Year") -> df
}
```

Zadanie 4.

```
df_sql_4 <- function(df1){

  Posts <- df1

  sqldf("SELECT
        Questions.Id,
        Questions.Title,
        BestAnswers.MaxScore,
        Posts.Score AS AcceptedScore,
        BestAnswers.MaxScore-Posts.Score AS Difference
        FROM (
        SELECT Id, ParentId, MAX(Score) AS MaxScore
        FROM Posts
        WHERE PostTypeId==2
        GROUP BY ParentId
        ) AS BestAnswers
        JOIN (
        SELECT *
        FROM Posts
        WHERE PostTypeId==1
        ) AS Questions ON Questions.Id=BestAnswers.ParentId
        JOIN Posts ON Questions.AcceptedAnswerId=Posts.Id
        WHERE Difference>50
        ORDER BY Difference DESC") -> df
}
```

Zadanie 5.

```
df_sql_5 <- function(df1, df2){
  Posts <- df1
  Comments <- df2

  sqldf("SELECT
        Posts.Title,
        CmtTotScr.CommentsTotalScore
        FROM (
        SELECT
        PostID,
        UserID,
        SUM(Score) AS CommentsTotalScore
        FROM Comments
        GROUP BY PostID, UserID
        ) AS CmtTotScr
        JOIN Posts ON Posts.ID=CmtTotScr.PostID AND Posts.OwnerUserId=CmtTotScr.UserID
        WHERE Posts.PostTypeId=1
        ORDER BY CmtTotScr.CommentsTotalScore DESC
        LIMIT 10") -> df
}
```

Zadanie 6.

```
df_sql_6 <- function(df1, df2, df3){
  Posts <- df1
  Badges <- df2
  Users <- df3

  sqldf("SELECT DISTINCT
        Users.Id,
        Users.DisplayName,
        Users.Reputation,
        Users.Age,
        Users.Location
        FROM (
        SELECT Name, UserID
        FROM Badges
        WHERE Name IN (
        SELECT Name
        FROM Badges
        WHERE Class=1
        GROUP BY Name
        HAVING COUNT(*) BETWEEN 2 AND 10
        )
        AND Class=1
        ) AS ValuableBadges
        JOIN Users ON ValuableBadges.UserId=Users.Id") -> df
}
```

Zadanie 7.

```
df_sql_7 <- function(df1, df2){
  Posts <- df1
  Votes <- df2

  sqldf("SELECT
        Posts.Title,
        VotesByAge2.OldVotes
        FROM Posts
        JOIN (
        SELECT
        PostId,
        MAX(CASE WHEN VoteDate = 'new' THEN Total ELSE 0 END) NewVotes,
        MAX(CASE WHEN VoteDate = 'old' THEN Total ELSE 0 END) OldVotes,
        SUM(Total) AS Votes
        FROM (
        SELECT
        PostId,
        CASE STRFTIME('%Y', CreationDate)
        WHEN '2017' THEN 'new'
        WHEN '2016' THEN 'new'
        ELSE 'old'
        END VoteDate,
        COUNT(*) AS Total
        FROM Votes
        WHERE VoteTypeId=2
        GROUP BY PostId, VoteDate
        ) AS VotesByAge
        GROUP BY VotesByAge.PostId
        HAVING NewVotes=0
        ) AS VotesByAge2 ON VotesByAge2.PostId=Posts.ID
        WHERE Posts.PostTypeId=1
        ORDER BY VotesByAge2.OldVotes DESC
        LIMIT 10") -> df
}
```

4.2 Bazowy R

Zadanie 1.

```
df_base_1 <- function(df1, df2){

Posts <- df1
Users <- df2

x <- Posts[Posts$PostTypeId == 1, c("OwnerUserId", "FavoriteCount")]
#wybor kolumn do aggregate'a
x <- na.omit(x)
#opuszczamy NA
x1 <- aggregate(x["FavoriteCount"], x["OwnerUserId"], max)
colnames(x1)[2] <- "MostFavoriteQuestionLikes"
#aggregate dla fukcji max
x2 <- aggregate(x["FavoriteCount"], x["OwnerUserId"], sum)
colnames(x2)[2] <- "FavoriteTotal"
#aggregate dla funkcji sum
Posts2 <- Posts[, c("OwnerUserId", "FavoriteCount", "Title")]
colnames(Posts2)[2:3] <- c("MostFavoriteQuestionLikes", "MostFavoriteQuestion")
#pomocniczy Posts do merge'a

x <- merge(Posts2, x2)
x <- merge(x, x1)
#merge po favCount
colnames(x)[1] <- "Id"
x <- merge(x, Users)
#merge po Id
x <- x[, c("DisplayName", "Age", "Location",
           "FavoriteTotal", "MostFavoriteQuestion", "MostFavoriteQuestionLikes")]
#wybor kolumn
x <- x[ order(x$FavoriteTotal, decreasing = TRUE),]
#sort
rownames(x) <- NULL
x <- head(x, 10)
#ta dam!
}
```

Zadanie 2.

```
df_base_2 <- function(df1){

x <- as.data.frame(
  table(Posts[ Posts$PostTypeId == 2 & Posts$Score > 0, "ParentId" ]),
  stringsAsFactors = FALSE)
#tworzymy df-a takiego jak w wywolaniu sqlowym,
#pomocniczego, gdzie zliczamy ilosc pozytywnie ocenionych odpowiedzi
#na dane pytanie (o danym Id oznaczonym ParentId)

colnames(x) <- c("Id", "PositiveAnswerCount")
#zmieniamy nazwy kolumn, przy czym ta opisujaca
#ParentId nazywamy po prostu Id, zeby przy merge'owaniu
#bylo latwiej (bo merge wybierze tylko powtarzajace sie Id)


x2 <- Posts[ Posts$PostTypeId == 1, c("Id", "Title") ]
#tworzymy drugiego df-a z postami, ktore sa pytaniami,
#i trzymamy ich id i tytuly

x <- merge(x2, x)
#merge'ujemy po kolumnie o tej samej nazwie - Id
x <- x[order(x$PositiveAnswerCount, decreasing = TRUE), ]
#sortujemy malejaca permutacja sortujaca
rownames(x) <- NULL
#zmieniamy numeracje indeksow (takich prawdziwych, nie kolumny w df-ie)
x <- head(x, 10)
#bierzemy pierwsze 10 - sqlowy LIMIT 10
#ta dam!
}
```

Zadanie 3.

```
df_base_3 <- function(df1, df2){
Posts <- df1
Votes <- df2
x <- Votes[Votes$VoteTypeId == 2,]
#filter
x1 <- as.data.frame(substring(x$CreationDate, 1, 4))
#year wyciagamy substringiem jako osobna ramke danych
x <- cbind(x, x1)
#ktora laczymy z caloscia
colnames(x)[7] <- "Year"
#zmianiamy nazwe
x <- aggregate(x$VoteTypeId == 2, x[c("Year", "PostId")], length)
#zliczamy po yera i postId, wczesniej filtrujac
colnames(x)[2:3] <- c("Id", "Count")
#zmiana nazw pod merge
x <- merge(x, Posts)
#merge po Id
x <- x[x$PostTypeId == 1,]
#filter
x2 <- aggregate(x["Count"], x["Year"], max)
#pomocnicza ramka z maxem z Count po Year
x <- merge(x2, x, by = c("Year", "Count"))
#merge po Year i Count
x <- x[, c("Title", "Year", "Count")]
#wybor kolumn
#ta dam!
}
```

Zadanie 4.

```
df_base_4 <- function(df1){

Posts <- df1

x <- aggregate(Posts$Score, Posts["ParentId"], max)
#aggregate wywala NA na ParentId, ktore sa tylko przy PostTypeId = 1,
#Id nie potrzebujemy, wiec mamy juz wszystko
colnames(x) <- c("Id", "MaxScore")
#zmiana nazw pod merge'a
x1 <- Posts[Posts$PostTypeId == 1, c("Id", "Title", "AcceptedAnswerId")]
#Posts pomocnicze, filtrowane, potrzebne kolumny (1. join), nazwy do merge'a
x <- merge(x1, x, by = "Id")
#merge po Id
x2 <- Posts[, c("Score", "Id")]
#drugi posts, (2. join).
colnames(x2)[2] <- "AcceptedAnswerId"
#zmiana nazwy do merge'a
x <- merge(x, x2)
#merge po AcceptedAnswerId
x3 <- as.data.frame(x$MaxScore - x$Score)
#tworzenie osobno kolumny Difference
x <- cbind(x, x3)
#dodanie kolumny Difference do df
colnames(x)[5:6] <- c("AcceptedScore", "Difference")
#zmiany nazw na odpowiednie
x <- x[ order(x$Difference, decreasing = TRUE),]
#sortowanie
x <- x[ x$Difference > 50,
        c("Id", "Title", "MaxScore", "AcceptedScore", "Difference")]
#wybór kolumn i filter
#ta dam!
}
```

Zadanie 5.

```
df_base_5 <- function(df1, df2){
Posts <- df1
Comments <- df2

x <- aggregate(Comments$Score, by = Comments[c("PostId", "UserId")], FUN = sum)
# sumujemy Score po PostId i UserId
colnames(x) <- c("Id", "OwnerUserId", "CommentsTotalScore")
#nazwy kolumn do merge'a i wymagane
x <- merge(x, Posts, by = c("Id", "OwnerUserId"))
#merge po Id i OwnerUserId z Posts
x <- x[x$PostTypeId == 1, c("Title", "CommentsTotalScore")]
#filtrowanie i wybor kolumn
x <- head(x[order(x$CommentsTotalScore, decreasing = TRUE),], 10)
#sortowanie i head
#ta dam!
}
```

Zadanie 6.

```
df_base_6 <- function(df1, df2, df3){
Posts <- df1
Badges <- df2
Users <- df3
x <- as.data.frame(table(Badges[ Badges$Class == 1, "Name"]),
                   stringsAsFactors = FALSE)
#zliczamy po Name ilosc przy Class == 1 w Badges i rzucamy do dfa
colnames(x) <- c("Name", "count")
#zmiana nazw na jakies ludzkie
x <- x[ x$count >= 2 & x$count <= 10, c("Name", "count")]
#wybor tych kolumn przy warunku
x <- merge(Badges, x)
#merge z Badges po Name
x <- x[ x$Class == 1, c("Name", "UserId")]
#wybieramy przy warunku kolumny
colnames(x)[2] = "Id"
#zmiana nazwy do merge'a
x <- merge(Users, x)
#merge z Users po Id
x <- x[, c("Id", "DisplayName", "Reputation", "Age", "Location")]
#wybor kolumn
x <- unique(x)
#wybor unikalnych wierszy

#ta dam!

}
```

Zadanie 7.

```
df_base_7 <- function(df1, df2){
Posts <- df1
Votes <- df2

x <- Votes[Votes$VoteTypeId == 2,]
x1 <- as.data.frame(as.integer(
  substring(x$CreationDate, 1, 4) == "2016" | substring(x$CreationDate, 1, 4) == "2017"))
x <- cbind(x, x1)
colnames(x)[7] <- "VoteDate"
x <- aggregate(x$VoteTypeId, x[c("PostId" ,"VoteDate")], length)
colnames(x)[c(1, 3)] <- c("Id", "Total")
#
x2 <- x["Total"] * x["VoteDate"]
x3 <- x["Total"] * (1 - x["VoteDate"])
x <- cbind(x, x2)
x <- cbind(x, x3)
colnames(x)[3:5] <- c("Votes" ,"NewVotes", "OldVotes")
x2 <- aggregate(x$NewVotes, x["Id"], max)
x3 <- aggregate(x$OldVotes, x["Id"], max)
x <- aggregate(x$Votes, x["Id"], sum)
x <- merge(x, x2, by = "Id")
x <- merge(x, x3, by = "Id")
colnames(x)[2:4] <- c("Votes", "NewVotes", "OldVotes")
## to cholerstwo to jeden summarize
x <- x[x$NewVotes == 0,]
x <- merge(x, Posts)
x <- x[x$PostTypeId == 1, c("Title", "OldVotes")]
x <- head(x[ order(x$OldVotes, decreasing = TRUE),], 10)
#ta dam!
}
```

4.3 data.table

Zadanie 1.

```
df_table_1 <- function(df1, df2){

Posts <- df1
Users <- df2

y <- Posts[PostTypeId == 1, .(OwnerUserId, FavoriteCount)]
#filtrujemy posts i wybieramy kolumny
y <- na.omit(y)
#opuszczamy NA
y <- y[, .(FavoriteTotal = sum(FavoriteCount),
           MostFavoriteQuestionLikes = max(FavoriteCount)),
  by = .(OwnerUserId)]
#tworzymy kolumny FT i MFQL robiac funkcje po OwnerUserId
Posts2 <- Posts[,.(OwnerUserId,
                   MostFavoriteQuestionLikes = FavoriteCount,
                   MostFavoriteQuestion = Title)]
#pomocniczy posts do merge'a
y <- Posts2[y, on = c("OwnerUserId", "MostFavoriteQuestionLikes")]
#merge
setnames(y, c("OwnerUserId"), c("Id"))
#ustawienie nazw kolumn do merga
y <- y[Users, on = "Id"][
#merge
  ,.(DisplayName, Age, Location, FavoriteTotal,
     MostFavoriteQuestion, MostFavoriteQuestionLikes)][
#wybieramy kolumny
  order(-FavoriteTotal)][
#sort
  1:10]
#head
#ta dam!
}
```

Zadanie 2.

```
df_table_2 <- function(df1){

Posts <- df1

#przetworzenie Posts do data.table

y <- Posts[PostTypeId == 2 & Score > 0, .N, by = "ParentId"]
# zlicz (.N) po ParentId przy spelnionym warunku
setnames(y, c("ParentId", "N"), c("Id", "PositiveAnswerCount"))
#zmieniamy nazwy kolumn, przy czym ta opisujaca ParentId
#nazywamy po prostu Id, zeby przy merge'owaniu bylo katwiej
#(bo merge wybierze tylko powtarzajace sie Id)

y2 <- Posts[PostTypeId == 1, c("Id", "Title")]
#tworzymy drugiego dt-a z postami, ktore sa pytaniami, i trzymamy ich id i tytuly

y <- y2[y, on = c("Id")][
#merge'ujemy po kolumnie o tej samej nazwie - Id
order(-PositiveAnswerCount)][
#bierzemy heada z y posortowanego po PAC malejaco
1:10]
#head
#ta dam!
}
```

Zadanie 3.

```r
df_table_3 <- function(df1, df2){
Posts <- df1
Votes <- df2
#Posts i Votes do dt
y <- Votes[VoteTypeId == 2, Year := substring(CreationDate, 1, 4)][
#warunek, kolumna Year jako cztery znaki
, .N, by = .(Year, PostId)]
#zliczanie po Year i PostId

setnames(y, c("N", "PostId"), c("Count", "Id"))
#rename do merge'a
y <- y[Posts, on = "Id"][
#merge po Id
PostTypeId == 1, .(Title, Year, Count, max(Count)), Year][
#przefiltrowny x z max z Count po Year i wybranymi columnami
V4 == Count, .(Title, Year, Count)]
#filtrowanie max(Count) = Count, wybor kolumn
#ta dam!
}
```

Zadanie 4.

```r
df_table_4 <- function(df1){

Posts <- df1
Posts <- setDT(Posts)
#Posts do dt-a
y <- Posts[PostTypeId == 2, .(ParentId, Score)][
#wybieram dwie kolumny, dla spelnionego warunku
, max(Score), by = ParentId]
#liczy maxa ze Score, po ParentId
data.table::setnames(y, c("ParentId", "V1"), c("Id", "MaxScore"))
#zmiana nazwy zgodnie z wytycznymi + Id pod merge'a

y2 <- Posts[PostTypeId == 1, .(Id, Title, AcceptedAnswerId)]
#wybieram trzy kolumny dla warunku
y3 <- Posts[, .(Score, Id)]
#a tu po prostu dwie kolumny
data.table::setnames(y3, "Id", "AcceptedAnswerId")
#zmiana nazwy pod merge'a
y <- y[y2, on = c("Id")][
#laczenie po Id
y3, on = "AcceptedAnswerId"][
#laczenie po AcceptedAnswerId
, Difference := MaxScore - Score ][
#dodanie kolumny przez przypisanie
order(-Difference)][
#sortowanie
Difference > 50][
#filtrowanie
,.(Id, Title, MaxScore, AcceptedScore = Score, Difference)]
#wybor interesujacych nas kolumn + zmiana nazwy na zadane
#ta dam!
}
```

Zadanie 5.

```
df_table_5 <- function(df1, df2){
Posts <- df1
Comments <- df2
Comments <- setDT(Comments)
#przetworzenie Comments do data.table
Comments <- Comments[, sum(Score), c("PostId", "UserId")]
#sumowanie Score po PostId i UserId z Comments, wybor tych 3 kolumn
data.table::setnames(Comments, c("PostId", "UserId", "V1"),
                     c("Id", "OwnerUserId", "CommentsTotalScore"))
#nazwanie ich odpowiednio, zeby merge dzialal,
Posts <- setDT(Posts)
#przetworzenie Posts do dt


y <- Posts[, c("Title", "Id", "OwnerUserId", "PostTypeId") ][
#wybor interesujacych nas kolumn
Comments, on = c("Id", "OwnerUserId")][
#meregujemy po OwnerUserId i Id
PostTypeId == 1, c("Title", "CommentsTotalScore"), ][
#wybieramy wiersze dla PostTypeId == 1, kolumny nas interesujace
order(-CommentsTotalScore)][
1:10]
#sort i head - 10 wynikow
#ta dam!
}
```

Zadanie 6.

```
df_table_6 <- function(df1, df2, df3){
Posts <- df1
Badges <- df2
Users <- df3
#Badges, Users do dt
y <- Badges[Class == 1, .N, Name][
#zliczanie po Name dla Class == 1
N >= 2 & N <= 10,]
#wybor kolumn z N spelniajacych warunek
y <- Badges[y, on = c("Name")][
#mwrge po Name
Class == 1, .(Id = UserId, Name)]
#wybor kolumn Name i UserId jako Id pod merga, plus filter Class == 1
y <- Users[y, on = c("Id")][
#merge po Id
, c("Id", "DisplayName", "Reputation", "Age", "Location")]
#wybĂłr interesujÄ...cych nas kolumn
y <- unique(y)
#wybor unikalnych rekordow
#ta dam!
}
```

Zadanie 7.

```
df_table_7 <- function(df1, df2){
Posts <- df1
Votes <- df2

Votes <- setDT(Votes)
y <- Votes[VoteTypeId == 2,][
, VoteDate := as.integer(substring(CreationDate, 1, 4)) ][
#wyciagamy i rzutujemy do integera rok, zeby nie bylo problemow ze zmianem typu kolumny
VoteDate == 2016 | VoteDate == 2017, VoteDate := 1,][
! VoteDate == 1, VoteDate := 0][
#uzupelniamy 0 i 1 dla New i Old, i tak tych nazw nie uzywamy
, .(Total = .N), by = .(PostId, VoteDate)][
#zliczamy w Total ilosc po PostId i VoteDate, i mamy VotesByAge
, .(Id = PostId, NewVotes = max(Total * VoteDate),
    OldVotes = max(Total * (1 - VoteDate)),
    Votes = sum(Total)), PostId][
# wybieramy kolumny, liczac oldV, newV i V po PostId
NewVotes == 0,][
#filter i mamy VotesByAge2
Posts, on = "Id"][
#merge z Posts po Id
PostTypeId == 1, .(Title, OldVotes)][
#filter i select
order(-OldVotes)][
1:10]
#sort i head
#ta dam!
}
```

4.4 dplyr

Zadanie 1.

```r
df_dplyr_1 <- function(df1, df2){

Posts <- df1
Users <- df2

Posts %>%
  select(OwnerUserId, FavoriteCount) %>%
#filtrowanie posts
  group_by(OwnerUserId) %>%
#wybieramy FavoriteCount do zliczania i OwnerUserId
  na.omit() -> z
#pozbywamy się NA - czyli jednoczesnie filtrujemy po PostTypeId

z1 <- summarise(z, FavoriteTotal = sum(FavoriteCount))
z2 <- summarise(z, MostFavoriteQuestionLikes = max(FavoriteCount))
#wyliczenie kolumn FT i MFQL
Posts2 <- select(Posts, OwnerUserId,
                 MostFavoriteQuestionLikes = FavoriteCount,
                 MostFavoriteQuestion = Title)
#Posts2 - pomocniczy posts do mergeowanie


z <- inner_join(Posts2, z2, by = c("OwnerUserId", "MostFavoriteQuestionLikes"))
z <- inner_join(z, z1)
#merge z nowymi kolumnami
z <- rename(z, Id = OwnerUserId)
z <- inner_join(z, Users)
#merge z users po Id

z %>%
  select(DisplayName, Age, Location, FavoriteTotal,
         MostFavoriteQuestion, MostFavoriteQuestionLikes) %>%
#wybor kolumn
  arrange(desc(FavoriteTotal)) %>%
#sort
  slice(1:10) -> z
#head
#ta dam!
}
```

Zadanie 2.

```r
df_dplyr_2 <- function(df1){

Posts <- df1
Posts %>%
filter(PostTypeId == 2, Score > 0) %>%
# filtrowanie rzedow spelniajacych warunki
select(ParentId) %>%
#wybor kolumny
group_by(ParentId) %>%
#grupowanie po argumentach z kolumny
summarise(PositiveAnswerCount = n()) %>%
#liczenie dla poszczegolnych argumentow
rename(Id = ParentId) -> z
#zmieniamy nazwy kolumn, przy czym ta opisujaca
#ParentId nazywamy po prostu Id, zeby przy merge'owaniu
#bylo latwiej (bo merge wybierze tylko powtarzajÄace sie Id)

Posts %>%
filter(PostTypeId == 1) %>%
#filtrowanie rzedow spelniajacych warunki
select(Id, Title) -> z2
#wybor kolumn do z2


z <- inner_join(z2, z)
#merge'ujemy po kolumnie o tej samej nazwie - Id
z <- slice(arrange(z, desc(PositiveAnswerCount)), 1:10)
#bierzemy heada z x posortowanego po PAC malejaco ("-")
#dplyr::all_equal(z, df1)
#ta dam!
}
```

Zadanie 3.

```r
df_dplyr_3 <- function(df1, df2){
Posts <- df1
Votes <- df2
Votes %>%
filter(VoteTypeId == 2) %>%
#filtrujemy
mutate(Year = substring(CreationDate, 1, 4)) %>%
#wyciagamy rok - pierwsze cztery znaki ze stringa
group_by(Year, Id = PostId) %>%
#rename do merge'a
summarize(Count = n()) -> z
#sumowanie po Year i Id

z <- inner_join(z, Posts)
#merge po Id
z %>%
filter(PostTypeId == 1) %>%
select(Title, Year, Count) %>%
group_by(Year) -> z
#przygotowanie do summarize

z1 <- summarize(z, Count = max(Count))
#max z Counta po roku
z <- inner_join(z, z1)
#merge po Yaer i Count

#all_equal(z, df1, convert = TRUE)
#ta dam!
}
```

Zadanie 4.

```
df_dplyr_4 <- function(df1){

Posts <- df1
Posts %>%
filter(PostTypeId == 2) %>%
select(ParentId, Score) %>%
#Score chcemy, ParentId do merge'a
group_by(ParentId) %>%
summarize(MaxScore = max(Score)) %>%
#liczymy MaxScore
na.omit() %>%
#zeby pozbyl sie nieznanych ParentId
rename(Id = ParentId) -> z1


Posts %>%
filter(PostTypeId == 1) %>%
select(Id, Title, AcceptedAnswerId) -> z2
#Id, Title - potrzebne/ Id, AcceptedAnswerId - merge

z3 <- select(Posts, Score, AcceptedAnswerId = Id)
#Potrzebujemy Score, merge po Id

z <- inner_join(z2, z1) #laczymy po Id
z <- inner_join(z, z3) #laczymy po AcceptedAnswerId

z %>%
mutate(Difference = MaxScore - Score) %>%
#dodajemy kolumne Difference
rename(AcceptedScore = Score) %>%
#Nazywamy Score odpowiednio
filter(Difference > 50) %>%
arrange(desc(Difference)) %>%
#sortowanko
select(Id, Title, MaxScore, AcceptedScore, Difference) -> z
#interesujace nas kolumny
#ta dam!
}
```

Zadanie 5.

```
df_dplyr_5 <- function(df1, df2){
Posts <- df1
Comments <- df2

Comments %>%
group_by(PostId, UserId) %>%
#grupowanie po PostId i UserId przed sumowaniem
summarize(CommentsTotalScore = sum(Score)) %>%
#sumowanie do z
rename(Id = PostId, OwnerUserId = UserId) -> z
#zmiana nazw kolumn, zeby merge nie glupial



Posts %>%
filter(PostTypeId == 1) %>%
#filtrowanie wierszy
select(Id, OwnerUserId, Title) -> z2
#wybĂłr interesujacych nas kolumn

inner_join(z2, z) %>%
#merge wzgledem juz przefiltrowanego z2 - wiec jednoczesnie "filtrujemy" z
select(Title, CommentsTotalScore) %>%
#wybor interesujacych nas kolumn
arrange(desc(CommentsTotalScore)) %>%
#sortowanie
slice(1:10) -> z
#wybor 10 pierwszych rekordow
#ta dam!
}
```

Zadanie 6.

```
df_dplyr_6 <- function(df1, df2, df3){
Posts <- df1
Badges <- df2
Users <- df3
Badges %>%
filter(Class == 1) %>%
#filtrujemy po Class
group_by(Name) %>%
#grupujemy po Name
summarize(count = n()) %>%
#zliczamy po Name
filter(count >= 2 & count <= 10) -> z
#bierzemy interesujÄ...ce nas

z <- inner_join(Badges, z)
#merge z Badges po Name

z %>%
filter(Class == 1) %>%
#filtrujemy znowu, bo doszlo nam przy mergu z Badges
select(Name, Id = UserId) -> z
#wybieramy co chcemy, plus zmieniamy nazwe do merga

z <- inner_join(Users, z)
#merge z Users po Id
z %>%
select(Id, DisplayName, Reputation, Age, Location) %>%
distinct() -> z
#wybieramy co chcemy, i rozne wiersze
#ta dam!
}
```

Zadanie 7.

```
df_dplyr_7 <- function(df1, df2){
Posts <- df1
Votes <- df2

Votes %>%
filter(VoteTypeId == 2) %>%
mutate(VoteDate = as.integer(
  substring(CreationDate, 1, 4) == "2016" | substring(CreationDate, 1, 4) == "2017")) %>%
#wyciagamy rok ze stringa i rzutujemy boola do integera,
#zeby miec 1 dla New i 0 dla Old
group_by(PostId, VoteDate) %>%
summarize(Total = n()) %>%
#zliczamy w Total ilosc po PostId i VoteDate, i mamy VotesByAge
group_by(Id = PostId) %>%
summarize(NewVotes = max(Total * VoteDate),
          OldVotes = max(Total * (1 - VoteDate)),
          Votes = sum(Total)) %>%
# wybieramy kolumny, liczac oldV, newV i V po PostId
filter(NewVotes == 0) %>%
#filter i mamy VotesByAge2
inner_join(Posts) %>%
#merge z Posts po Id
filter(PostTypeId == 1) %>%
select(Title, OldVotes) %>%
#filter i select
arrange(desc(OldVotes)) %>%
slice(1:10) -> z
#sort i head
#ta dam!
}
```

5 Wywołania

Zadanie 1.

```
dfsql <- df_sql_1(Posts, Users)
dfbase <- df_base_1(Posts, Users)
dftable <- df_table_1(PostsDT, UsersDT)
dfdplyr <- df_dplyr_1(Posts, Users)

all_equal(dfsql, dfbase, convert = TRUE)

## [1] TRUE

all_equal(dfsql, dftable, convert = TRUE)

## [1] TRUE

all_equal(dfsql, dfdplyr, convert = TRUE)

## [1] TRUE
```

Zadanie 2.

```
dfsql <- df_sql_2(Posts)
dfbase <- df_base_2(Posts)
dftable <- df_table_2(PostsDT)
dfdplyr <- df_dplyr_2(Posts)

all_equal(dfsql, dfbase, convert = TRUE)

## [1] TRUE

all_equal(dfsql, dftable, convert = TRUE)

## [1] TRUE

all_equal(dfsql, dfdplyr, convert = TRUE)

## [1] TRUE
```

Zadanie 3.

```
dfsql <- df_sql_3(Posts, Votes)
dfbase <- df_base_3(Posts, Votes)
dftable <- df_table_3(PostsDT, VotesDT)
dfdplyr <- df_dplyr_3(Posts, Votes)

all_equal(dfsql, dfbase, convert = TRUE)

## Warning: Column ‘Year‘ joining character vector and factor, coercing into character
vector

## [1] TRUE

all_equal(dfsql, dftable, convert = TRUE)

## [1] TRUE

all_equal(dfsql, dfdplyr, convert = TRUE)

## [1] TRUE
```

Zadanie 4.

```
dfsql <- df_sql_4(Posts)
dfbase <- df_base_4(Posts)
dftable <- df_table_4(PostsDT)
dfdplyr <- df_dplyr_4(Posts)

all_equal(dfsql, dfbase, convert = TRUE)

## [1] TRUE

all_equal(dfsql, dftable, convert = TRUE)

## [1] TRUE

all_equal(dfsql, dfdplyr, convert = TRUE)

## [1] TRUE
```

Zadanie 5.

```
dfsql <- df_sql_5(Posts, Comments)
dfbase <- df_base_5(Posts,Comments)
dftable <- df_table_5(PostsDT, CommentsDT)
dfdplyr <- df_dplyr_5(Posts, Comments)

all_equal(dfsql, dfbase, convert = TRUE)

## [1] TRUE
```

```
all_equal(dfsql, dftable, convert = TRUE)

## [1] TRUE

all_equal(dfsql, dfdplyr, convert = TRUE)

## [1] TRUE
```

Zadanie 6.

```
dfsql <- df_sql_6(Posts, Badges, Users)
dfbase <- df_base_6(Posts, Badges, Users)
dftable <- df_table_6(PostsDT, BadgesDT, UsersDT)
dfdplyr <- df_dplyr_6(Posts, Badges, Users)

all_equal(dfsql, dfbase, convert = TRUE)

## [1] TRUE

all_equal(dfsql, dftable, convert = TRUE)

## [1] TRUE

all_equal(dfsql, dfdplyr, convert = TRUE)

## [1] TRUE
```

Zadanie 7.

```
dfsql <- df_sql_7(Posts, Votes)
dfbase <- df_base_7(Posts, Votes)
dftable <- df_table_7(PostsDT, VotesDT)
dfdplyr <- df_dplyr_7(Posts, Votes)

all_equal(dfsql, dfbase, convert = TRUE)

## [1] TRUE

all_equal(dfsql, dftable, convert = TRUE)

## [1] TRUE

all_equal(dfsql, dfdplyr, convert = TRUE)

## [1] TRUE
```

6 Benchmarki

Zadanie 1.

```
microbenchmark::microbenchmark(
  sqldf1 = df_sql_1(Posts, Users),
  base1= df_base_1(Posts, Users),
  dplyr1 = df_dplyr_1(Posts, Users),
  table1 = df_table_1(PostsDT, UsersDT),
  times = 100
)

## Unit: milliseconds
##     expr       min        lq      mean    median        uq      max neval
##   sqldf1 289.77723 298.01005 318.83126 306.83528 318.97415 480.0738   100
##    base1 193.66482 207.70421 220.10891 214.77026 224.63528 392.6240   100
##   dplyr1  42.80205  46.77067  57.67889  50.17518  61.37333 369.2181   100
##   table1  31.74605  35.36574  47.09301  38.03077  51.09456 449.5151   100
```

Zadanie 2.

```
microbenchmark::microbenchmark(
  sqldf2 = df_sql_2(Posts),
  base2= df_base_2(Posts),
  dplyr2 = df_dplyr_2(Posts),
  table2 = df_table_2(PostsDT),
  times = 100
)

## Unit: milliseconds
##     expr       min        lq      mean    median        uq       max neval
##   sqldf2 203.79815 208.86338 217.63931 212.35733 219.98133 305.08841   100
##    base2  75.68820  79.57005  88.76174  84.69251  93.62441 200.59980   100
##   dplyr2  42.48410  44.61210  50.97266  45.97210  56.62810 155.06585   100
##   table2  19.55651  20.70195  23.75588  22.38749  23.70769  38.80985   100
```

Zadanie 3.

```
microbenchmark::microbenchmark(
  sqldf3 = df_sql_3(Posts, Votes),
  base3 = df_base_3(Posts, Votes),
  dplyr3 = df_dplyr_3(Posts, Votes),
  table3 = df_table_3(PostsDT, VotesDT),
  times = 100
)

## Unit: milliseconds
##     expr       min        lq      mean    median        uq       max neval
##   sqldf3 1074.4718 1088.3120 1115.7243 1096.2441 1115.9807 1390.4078   100
##    base3 2758.7528 2860.8258 2926.8451 2904.7768 2991.2098 3315.5323   100
##   dplyr3  194.6359  227.2285  246.0361  237.1696  260.0398  402.3159   100
##   table3  140.5530  154.5719  179.8587  174.5383  181.3793  379.8023   100
```

Zadanie 4.

```
microbenchmark::microbenchmark(
  sqldf4 = df_sql_4(Posts),
  base4 = df_base_4(Posts),
  dplyr4 = df_dplyr_4(Posts),
  table4 = df_table_4(PostsDT),
  times = 100
)
```

```
## Unit: milliseconds
##    expr       min        lq      mean    median        uq       max neval
##  sqldf4 267.34646 273.20554 278.49090 275.70298 280.55508 318.18626   100
##   base4 301.61600 311.05682 319.52254 315.09354 331.08533 351.08267   100
##  dplyr4  57.09128  59.55692  66.30864  61.24349  65.36328 196.08492   100
##  table4  41.43631  43.85580  46.42348  45.95713  46.95610  65.92739   100
```

Zadanie 5.

```
microbenchmark::microbenchmark(
  sqldf5 = df_sql_5(Posts, Comments),
  base5 = df_base_5(Posts, Comments),
  dplyr5 = df_dplyr_5(Posts, Comments),
  table5 = df_table_5(PostsDT, CommentsDT),
  times = 100
)
```

```
## Unit: milliseconds
##    expr        min         lq       mean     median         uq       max
##  sqldf5  477.65456  488.95179  498.33480  492.79036  499.24656  566.9912
##   base5 2879.91549 2959.47713 3015.92572 2994.60000 3037.79364 3375.4244
##  dplyr5  190.50257  231.45703  245.35040  237.33969  247.38872  426.3918
##  table5   38.98338   45.48656   47.79735   47.06277   48.33518  119.9028
##  neval
##    100
##    100
##    100
##    100
```

Zadanie 6.

```
microbenchmark::microbenchmark(
  sqldf6 = df_sql_6(Posts, Badges, Users),
  base6 = df_base_6(Posts, Badges, Users),
  dplyr6 = df_dplyr_6(Posts, Badges, Users),
  table6 = df_table_6(PostsDT, BadgesDT, UsersDT),
  times = 100
)
```

```
## Unit: milliseconds
##    expr        min        lq      mean    median        uq       max neval
##  sqldf6 215.925334 222.63672 228.20084 226.13949 230.06667 287.99672   100
##   base6  10.154667  11.01887  12.33206  11.31138  11.69128  44.04185   100
##  dplyr6   9.748513  10.69005  12.80697  11.15590  11.70195  87.17251   100
##  table6  14.625231  17.11364  19.32332  18.49949  19.56964  80.87221   100
```

Zadanie 7.

```
microbenchmark::microbenchmark(
  sqldf7 = df_sql_7(Posts, Votes),
  base7 = df_base_7(Posts, Votes),
  dplyr7 = df_dplyr_7(Posts, Votes),
  table7 = df_table_7(PostsDT, VotesDT),
  times = 100
)

## Unit: milliseconds
##    expr       min        lq      mean    median        uq       max neval
##  sqldf7 1034.1670 1056.8287 1127.3434 1076.0792 1147.0521 1640.5058   100
##   base7 3798.1932 3911.7009 4138.1524 4007.6722 4186.5061 5977.6501   100
##  dplyr7 1406.5235 1527.0669 1663.0423 1600.0379 1709.8031 2545.7945   100
##  table7  416.3819  460.5411  500.0493  479.7902  511.8677  875.9549   100
```