

Praca Domowa numer 1

15 kwietnia 2019

Jakub Wiśniewski

0 Przygotowanie do pracy

Przed przystąpieniem do pracy pobieram wymagane pakiety oraz ramki danych

```
library(sqldf)
library(dplyr)
library(data.table)
library(microbenchmark)

Badges <- read.csv("Badges.csv")
Comments <- read.csv("Comments.csv")
PostLinks <- read.csv("PostLinks.csv")
Posts <- read.csv("Posts.csv")
Tags <- read.csv("Tags.csv")
Users <- read.csv("Users.csv")
Votes <- read.csv("Votes.csv")

options(stringsAsFactors = FALSE)
```

W interpretacji pytań będę się posługiwał: <https://meta.stackexchange.com/questions/2677/database-schema-documentation-for-the-public-data-dump-and-sede>

1 Zadanie 1

1.1 Zapytanie SQL

Wyberzmy dziesięć najpopularniejszych pytań (jeśli PostTypeId ==1, to jest to pytanie), liczby favorites jakie dostały i informacje o ich postujących. Chcemy, żeby było uszeregowane od ilości sumy favorites, jakie dostał użytkownik kiedykolwiek.

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

  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") -> output
return(output)
}

```

1.2 Base R

```

df_base_1 <- function(Posts, Users) {

# wpięrw łączę Posts i Users
A <- merge( Posts, Users , by.x = "OwnerUserId", by.y = "Id")

#usuwam NA
A$FavoriteCount[is.na(A$FavoriteCount)] <- 0

# wykonuję operacje na grupach
B <- aggregate(A$FavoriteCount, by= list(A$OwnerUserId), sum )
C <- aggregate(A$FavoriteCount, by = list(A$OwnerUserId), max)

# porządkuję nazwy
colnames(B) <- c("OwnerUserId", "FavoriteTotal")
colnames(C) <- c("OwnerUserId", "MostFavoriteQuestionLikes")

# następnie merguje poprzednie ramki z wynikami oraz jeszcze wcześniejsze ramki
D <- merge(B, C, by.x = "OwnerUserId", by.y ="OwnerUserId" )

E <- merge(A , D , by.x = c("OwnerUserId", "FavoriteCount"),
           by.y =c("OwnerUserId", "MostFavoriteQuestionLikes"))
colnames(E)[2] <- "MostFavoriteQuestionLikes"

# wybieram pytania i interesujące mnie kolumny
G <- E[E$PostTypeId == 1, c("DisplayName", "Age", "Location",
                           "FavoriteTotal", "Title", "MostFavoriteQuestionLikes")]

# Formatuje wyniki
G <- G[order(-G$FavoriteTotal),]
colnames(G)[5] <- "MostFavoriteQuestion"
H <- head(G , n= 10)

rownames(H) <- NULL

# Zmieniam, żeby były kompatybilne z all_equals
H$MostFavoriteQuestion <- as.character(H$MostFavoriteQuestion)
H$FavoriteTotal <- as.integer(H$FavoriteTotal)
H$MostFavoriteQuestionLikes <- as.integer(H$MostFavoriteQuestionLikes)
return(H)
}

```

1.3 Dplyr

```
df_dplyr_1 <- function(Posts, Users) {  
  # wpieryw wykonuję joina  
  A <- inner_join(Posts, Users , by = c("OwnerUserId" = "Id" ))  
  
  # następnie wybieram pytania i wykonuję operacje na grupach  
  B <- A %>% filter(PostTypeId == 1) %>% group_by(OwnerUserId) %>%  
    summarise_at("FavoriteCount",  
                 list(FavoriteTotal = sum,  
                      MostFavoriteQuestionLikes = max), na.rm = TRUE)  
  
  # wykonuję joina z wcześniejszą tabelą a następnie porządkuję i wykonuję operacje  
  # by wynik był kompatybilny z zapytaniem SQL  
  C <- inner_join(B , A, by = c("OwnerUserId" = "OwnerUserId",  
                                "MostFavoriteQuestionLikes" = "FavoriteCount")) %>%  
    rename(MostFavoriteQuestion = Title ) %>%  
    select(DisplayName, Age, Location, FavoriteTotal,  
           MostFavoriteQuestion, MostFavoriteQuestionLikes) %>%  
    arrange(desc(FavoriteTotal)) %>% slice(1:10) %>%  
    mutate(MostFavoriteQuestionLikes = as.integer(MostFavoriteQuestionLikes)) %>%  
    mutate(MostFavoriteQuestion = as.character(MostFavoriteQuestion))  
  return(C)  
}
```

1.4 Data.table

```
df_table_1 <- function(Posts, Users) {  
  
  # Zamieniam data.frame na data.table  
  Utable <- as.data.table(Users)  
  Ptable <- as.data.table(Posts)  
  
  # przygotowuję do joina poprzez posortowanie i oznaczenie jako posortowane  
  setkey(Utable, Id)  
  setkey(Ptable, OwnerUserId)  
  
  # wybieram pytania i odsiewam NA  
  A <- Ptable[Utable][PostTypeId == 1][!is.na(FavoriteCount)]  
  
  # wykonuję operacje na grupach  
  B <- A[, .(DisplayName, Age, Location, FavoriteTotal = sum(FavoriteCount),  
            MostFavoriteQuestion = Title,  
            FavoriteCount,  
            MostFavoriteQuestionLikes = max(FavoriteCount)),  
          by = .(OwnerUserId)]  
  
  #Sortuję by dostać interesujący mnie wynik  
  C <- setorder(B, -FavoriteTotal, -FavoriteCount)
```

```
# wyświetlam tylko najwyższy wynik w grupie
D <- C[, head(.SD, 1), by= .(OwnerUserId)]

#biore interesujące mnie kolumny
D <- D[, .(DisplayName, Age, Location, FavoriteTotal,
           MostFavoriteQuestion = as.character(MostFavoriteQuestion),
           MostFavoriteQuestionLikes)]

E <- D[1:10]
return(E)
}
```

1.5 Testy i benchmarki

```
dplyr::all_equal(df_sql_1(Posts, Users), df_base_1(Posts, Users))

## [1] TRUE

dplyr::all_equal(df_sql_1(Posts, Users), df_dplyr_1(Posts, Users))

## [1] TRUE

dplyr::all_equal(df_sql_1(Posts, Users), df_table_1(Posts, Users))

## [1] TRUE

microbenchmark::microbenchmark(
  times = 10,
  sqldf_1 = df_sql_1(Posts, Users),
  base_1 = df_base_1(Posts, Users),
  dplyr_1 = df_dplyr_1(Posts, Users),
  data.table_1 = df_table_1(Posts, Users))

## Unit: milliseconds
##      expr      min       lq      mean    median      uq      max
##  sqldf_1 460.22880 482.29557 551.2079 556.46454 588.28124 663.6288
##   base_1 769.83042 787.96903 835.4685 814.62914 859.93818 973.0645
##  dplyr_1  69.85581  93.69043 125.3976 118.29101 164.61139 183.2951
## data.table_1 51.69714 57.75697  69.4477  64.89257  81.34886 105.2565
## neval
##    10
##    10
##    10
##    10
```

2 Zadanie 2

2.1 Zapytanie SQL

Wybermy te pytania, które miały najwięcej pozytywnych odpowiedzi.

```
df_sql_2 <- function(Posts) {
  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") -> df1
  return(df1)
}
```

2.2 Bazowy R

```
# wybieram selecta w nawiasach
df_base_2 <- function(Posts) {
  Posts2 <- as.data.frame(table(Posts[Posts$PostTypeId == 2 & Posts$Score > 0 ,
                                "ParentId"]), stringsAsFactors = FALSE)

  # przyporządkowuję nazwy w kolumnach
  colnames(Posts2) <- c("ParentId", "PositiveAnswerCount")
  lewy <- Posts[c("Id", "Title")]

  # łączę
  x <- merge(x= lewy, y=Posts2, by.x = 'Id', by.y = 'ParentId', all.x = TRUE, all.y = TRUE)

  #sortuję malejąco oraz wybieram 10 pierwszych
  x <- head(x[order(-x$PositiveAnswerCount),], n = 10)
  rownames(x) <- NULL
  return(x)
}
```

2.3 Dplyr

```
df_dplyr_2 <- function(Posts) {
  # nakładam filtry, biorę interesującą mnie grupę, liczę poszczególne wystąpienia i grupuję
  x <- Posts %>% filter(PostTypeId == 2 & Score > 0 )
  x<- x %>% select(ParentId) %>% count(ParentId) %>% group_by(ParentId)

  # zmieniam nazwy
  Posts2 <- x
}
```

```

colnames(Posts2)[2] <- "PositiveAnswerCount"

# przygotowuję się do złączenia
x <- Posts %>% select(Id, Title)

# łączę Id i Title z naszym Posts2
xy <- inner_join(x, Posts2, by = c("Id" = "ParentId"))

# sortuję po liczbie odpowiedzi i wybieram 10 pierwszych
xy <- xy %>% arrange(desc(PositiveAnswerCount))
xy <- slice(xy, 1:10)

return(xy)
}

```

2.4 Data.table

```

df_table_2 <- function(Posts) {

# zamieniam na data table i wykonuję przekształcenie do Posts2
Posts1 <- as.data.table(Posts)
Posts2 <- Posts1[PostTypeId ==2 & Score > 0, .N, by = ParentId ]
setnames(Posts2,"N", "PositiveAnswerCount" )

# wybieram kolumny
Posts1 <- Posts1[,c("Id", "Title")]

# i sortuję Posts1 po Id oraz Posts2 po ParentId
setkey(Posts1, Id)
setkey(Posts2, ParentId)

# dzięki wcześniejszej operacji wykonuję Joina
Posts1 <- Posts1[Posts2, nomatch= 0]

# sortuję malejąco i wybieram 10 pierwszych
setorder(Posts1, -PositiveAnswerCount)
Posts1 <- Posts1[1:10]

return(Posts1)
}

```

2.5 Testy i benchmarki

```

dplyr::all_equal(df_sql_2(Posts), df_dplyr_2(Posts))

## [1] TRUE

dplyr::all_equal(df_sql_2(Posts), df_base_2(Posts))

```

```
## [1] TRUE

dplyr::all_equal(df_sql_2(Posts), df_table_2(Posts))

## [1] TRUE

microbenchmark::microbenchmark(
  times = 10,
  sqldf_2 = df_sql_2(Posts),
  base_2 = df_base_2(Posts),
  dplyr_2 = df_dplyr_2(Posts),
  data.table_2 = df_table_2(Posts))

## Unit: milliseconds
##      expr      min       lq      mean     median        uq       max
##  sqldf_2 326.04451 326.89455 335.41555 328.40330 350.80264 355.2858
##  base_2  175.71380 178.68627 187.93066 184.05290 197.50239 205.4558
##  dplyr_2   64.62165  65.99353  80.02720  77.39374  95.57701 100.9336
## data.table_2 11.91526 12.01898 20.44176 12.47516 37.21216 39.8105
## neval
##      10
##      10
##      10
##      10
```

3 Zadanie 3

3.1 Zapytanie SQL

Wybierzmy pytania z największą liczbą pozytywnych głosów w danym roku.

```
df_sql_3 <- function (Posts, Votes) {
  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") -> df1
  return(df1)
}
```

3.2 Base R

```
df_base_3 <- function( Posts, Votes) {  
  
  # wybieram PostId z tych głosów, które były pozytywne  
  temp <- as.data.frame(Votes[Votes$VoteTypeId == 2, "PostId"], stringsAsFactors = FALSE)  
  colnames(temp)[1] <- "PostId"  
  
  # wybieram rok  
  temp$Year <- substring(Votes[Votes$VoteTypeId ==2 , "CreationDate"], 1, 4)  
  
  # zliczam wystąpienia w grupach  
  temp <- aggregate(temp$PostId, by = temp[,c("PostId", "Year")] ,FUN = length )  
  
  # porządkuję  
  colnames(temp) <- c("PostId", "Year", "Count")  
  UpvotesPerYear <- temp  
  
  # wykonuję joina na pytaniach  
  temp <- merge(Posts[Posts$PostTypeId == 1, ], UpvotesPerYear, by.x = "Id", by.y = "PostId")  
  
  # wybieram maksymalne wartości Counta z danego roku  
  temp1 <- aggregate(temp$Count, by= list(temp$Year), max)  
  colnames(temp1) <- c("Year", "Count")  
  
  # i dołączam do wcześniejszej by dołączyć do wyników tytuł  
  temp1 <- merge(temp, temp1, by.x = c("Year", "Count"), by.y = c("Year", "Count") )  
  
  # porządkuję  
  result <- temp1[,c("Title", "Year", "Count")]  
  return(result)  
}
```

3.3 Dplyr

```
df_dplyr_3 <- function(Posts, Votes) {  
  
  # wybieram rok używając funkcji data.table, bo jest szybsza od Base R  
  UpVotesPerYear <- Votes %>% mutate(Year = year(CreationDate)) %>%  
    # wybieram pozytywne głosy, grupuję i zliczam wystąpienia w grupach  
    filter(VoteTypeId == 2) %>% group_by(PostId, Year) %>%  
    count() %>% rename(Count = n) %>% select(PostId, Count, Year)  
  
  # joinuję z Posts i wybieram pytania, następnie zliczam maksymalne  
  # wartości dla danej grupy  
  A <- inner_join(Posts, UpVotesPerYear, by = c("Id" = "PostId")) %>%  
    filter(PostTypeId == 1) %>% group_by(Year) %>%  
    summarise(Count = max(Count))  
}
```



```

# joinuję, ale tym razem zależy mi na tytule...
B <- inner_join(Posts, UpVotesPerYear, by = c("Id" = "PostId")) %>%
  filter(PostTypeId == 1) %>% group_by(Year) %>%
  select(Title, Year, Count)

# który podpinam tutaj.
C <- inner_join(A,B , by = c("Year" = "Year" , "Count" = "Count")) %>%
  select(Title, Year, Count) %>% mutate(Year = as.character(Year))%>%
  as.data.frame()

# na koniec przekształcam aby było kompatybilne z zapytaniem SQL
C$Count <- as.integer(C$Count)
return(C)
}

```

3.4 Data.table

```

df_table_3 <- function(Posts, Votes) {

# Zmieniam na data.table
Ptable <- as.data.table(Posts)
Vtable <- as.data.table(Votes)

# Year staje się integerem
A <- Vtable[, Year := as.integer(year(CreationDate)),]

# wybieram pozytywne głosy i zliczam ich wystąpienia w grupach
UpVotesPerYear <- A[VoteTypeId == 2 , .(Count = .N), by = .(PostId, Year)]

# przygotowuje do joina
setkey(Ptable, Id)
setkey(UpVotesPerYear, PostId)

# joinuję i wybieram pytania
B <- Ptable[UpVotesPerYear][PostTypeId == 1]

# wybieram maksymalne wartości dla danego roku
C <- B[, .(Title = Title[which.max(Count)], Count = max(Count)), by = .(Year) ]

# na koniec dbam aby było kompatybilne z zapytaniem SQL
D <- C[, .(Title, Year = as.character(Year), Count)]

return(D)
}

```

3.5 Testy i benchmarki

```

dplyr::all_equal(df_sql_3(Posts, Votes), df_base_3(Posts, Votes))

## [1] TRUE

```

```

dplyr::all_equal(df_sql_3(Posts, Votes), df_dplyr_3(Posts, Votes))

## [1] TRUE

dplyr::all_equal(df_sql_3(Posts, Votes), df_table_3(Posts, Votes))

## [1] TRUE

microbenchmark::microbenchmark(
  times = 10,
  sqldf_3 = df_sql_3(Posts, Votes),
  base_3 = df_base_3(Posts, Votes),
  dplyr_3 = df_dplyr_3(Posts, Votes),
  data.table_3 = df_table_3(Posts, Votes))

## Unit: seconds
##      expr      min       lq      mean   median      uq      max  neval
##  sqldf_3 1.232054 1.233785 1.247816 1.242623 1.257557 1.283920     10
##  base_3   1.600670 1.625825 1.735825 1.725071 1.843020 1.899825     10
##  dplyr_3  3.896014 3.953441 4.034868 4.003698 4.063355 4.299343     10
## data.table_3 3.527179 3.637027 3.772625 3.810035 3.895526 3.954041     10

```

4 Zadanie 4

4.1 Zapytanie SQL

```

df_sql_4 <- function(Posts){
  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") -> df1
  return (df1)
}

```

4.2 Base R

```
df_base_4 <- function(Posts) {  
  
  # zliczam maksymalne Scory na grup ParentId, bez NA  
  A <- aggregate(Posts$Score , by = list(Posts$ParentId), max, na.rm= TRUE)  
  colnames(A) <- c("ParentId", "MaxScore")  
  
  # wybieram odpowiedzi z postów i interesujące mnie kolumny  
  B <- Posts[Posts$PostTypeId == 2, c("Id" ,"ParentId", "Score") ]  
  
  # łączę  
  C <- merge(B, A , by.x = c("ParentId" ), by.y = c("ParentId") )  
  
  # wybieram by się nie powtarzały  
  C <- C[C$Score == C$MaxScore ,]  
  
  # przypisuję  
  BestAnswers <- C[, c("Id" ,"ParentId", "MaxScore")]  
  
  # wybieram pytania z postów  
  Questions <- Posts[Posts$PostTypeId == 1,]  
  
  # łączę bez duplikatów  
  D <- merge(Questions, BestAnswers , by.x = "Id", by.y = "ParentId", no.dups = TRUE)  
  E <- merge(Posts, D, by.x = "Id", by.y = "AcceptedAnswerId", no.dups = TRUE)  
  
  # Liczę różnicę  
  E$Difference <- E$MaxScore - E$Score.x  
  
  # sortuję i wybieram te z różnicą ponad 50  
  E <- E[order(-E$Difference), ]  
  E <- E[E$Difference > 50,]  
  
  #porządkuję  
  E <- E[, c("Id", "Title.y", "MaxScore", "Score.x" , "Difference")]  
  colnames(E) <- c("Id", "Title", "MaxScore", "AcceptedScore", "Difference")  
  rownames(E) <- NULL  
  return(E)  
}
```

4.3 Dplyr

```
df_dplyr_4 <- function(Posts){  
  
  # wybieram odpowiedzi, zliczam maksymalne wartości dla grup, łączę z postami i  
  # wybieram interesujące mnie kolumny  
  BestAnswers <- Posts %>% filter(PostTypeId == 2) %>% group_by(ParentId) %>%  
    summarise(MaxScore = max(Score)) %>%
```

```

    inner_join(Posts, by = c("ParentId" = "ParentId",
                             "MaxScore" = "Score")) %>%
    select(Id, ParentId, MaxScore)

# wybieram pytania z postów
Questions <- Posts %>% filter(PostTypeId == 1)

# joinuję
A <- inner_join(Questions, BestAnswers , by = c("Id" = "ParentId"))

# następnie znów łączę, porządkuję, liczę różnicę i zmieniam kolumny by były
# kompatybilne z zapytaniem SQL
B <- inner_join(Posts, A, by = c("Id" = "AcceptedAnswerId") ) %>%
  rename(Title = Title.y, AcceptedScore = Score.x) %>%
  select(Id, Title, MaxScore, AcceptedScore) %>%
  mutate(Difference = MaxScore - AcceptedScore) %>%
  filter(Difference > 50) %>% arrange(desc(Difference)) %>%
  mutate(MaxScore = as.integer(MaxScore)) %>%
  mutate(Difference = as.integer(Difference)) %>% as.data.frame()
  return(B)
}

```

4.4 Data.table

```

df_table_4 <- function(Posts) {

# zmieniam na data.table
Ptable <- data.table(Posts)

# wybieram te odpowiedzi, które mają maksymalną ilość Score
BestAnswers <- Ptable[PostTypeId == 2, .(Id = Id[which.max(Score)],
    MaxScore = max(Score) ), by = ParentId ]

# wybieram pytania
Questions <- Ptable[PostTypeId == 1,,]

# sortuję przygotowując do joina
setkey(Questions, Id)
setkey(BestAnswers, ParentId)

# inner join
B <- Questions[BestAnswers, nomatch = 0 ]

# znów sortuję do joina
setkey(Ptable, Id)
setkey(B, AcceptedAnswerId)

# łączę
C <- B[Ptable]

```

```
# porządkuję, liczę różnicę i sortuję
D <- C[,.( Id, Title, MaxScore = as.integer(MaxScore), AcceptedScore = i.Score,
          Difference = MaxScore - i.Score) ][Difference > 50][order(-Difference)]
return(D)
}
```

4.5 Testy i benchmarki

```
#Testy, które mi nie przechodzą
dplyr::all_equal(df_sql_4(Posts), df_base_4(Posts))

## [1] "Rows in x but not y: 8, 7, 6, 5, 4, 3, 2, 1. Rows in y but not x: 8, 7, 6, 5, 4, 3, 2, 1. "

# ale gdy biorę zwyczajnie all.equals
all.equal(df_sql_4(Posts), df_base_4(Posts))

## [1] "Component \"Id\": Mean relative difference: 0.001017897"

#Tak samo tu:
dplyr::all_equal(df_sql_4(Posts), df_dplyr_4(Posts))

## [1] "Rows in x but not y: 8, 7, 6, 5, 4, 3, 2, 1. Rows in y but not x: 8, 7, 6, 5, 4, 3, 2, 1. "

all.equal(df_sql_4(Posts), df_dplyr_4(Posts))

## [1] "Component \"Id\": Mean relative difference: 0.001017897"

# nie wiem skąd wzięta się taka rozbieżność między Id.
# dodatkowo poza Id są takie same.
dplyr::all_equal(df_sql_4(Posts)[2:5], df_dplyr_4(Posts)[2:5])

## [1] TRUE

#byłbym wdzięczny za wszelkie podpowiedzi.

# ten przechodzi
dplyr::all_equal(df_sql_4(Posts), df_table_4(Posts))

## [1] TRUE

microbenchmark::microbenchmark(
  times = 10,
  sqldf_4 = df_sql_4(Posts),
  base_4 = df_base_4(Posts),
  dplyr_4 = df_dplyr_4(Posts),
  data.table_4 = df_table_4(Posts))

## Unit: milliseconds
##      expr      min       lq      mean  median       uq      max
##  sqldf_4 373.87584 379.12493 384.4607 385.3874 388.8945 394.7470
##   base_4 346.45053 352.15416 376.7609 357.4581 371.3204 504.5035
##  dplyr_4  92.27279 101.99245 117.2361 104.2701 147.0479 157.0702
## data.table_4 71.74675 84.80805 123.8721 126.9793 162.1028 170.8125
##  neval
##    10
##    10
##    10
##    10
```

5 Zadanie 5

5.1 Zapytanie SQL

Wyberzmy 10 pytań, których suma głosów dla odpowiedzi była największa.

```
df_sql_5 <- function(Posts, Comments){  
  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") -> df1  
  return(df1)  
}
```

5.2 Bazowy R

```
df_base_5 <- function(Posts, Comments){  
  
  # liczę sumę dla wyszczególnionych grup  
  CmtTotScr <- aggregate(Comments$Score , by = list(Comments$PostId, Comments$UserId), FUN=sum)  
  
  colnames(CmtTotScr) <- c("PostId", "UserId", "CommentsTotalScore")  
  
  # łączę z Posts  
  C <- merge(Posts, CmtTotScr, by.x = c("Id", "OwnerUserId") , by.y = cbind("PostId","UserId"))  
  
  # wybieram pytania z posts oraz interesujące nas kolumny  
  C <- C[C$PostTypeId == 1 , c("Title","CommentsTotalScore")]  
  
  # sortuję  
  C <- C[order(-C$CommentsTotalScore),]  
  
  # porządkuję  
  rownames(C) <- NULL  
  df1 <- head(C, n=10)  
  return(df1)  
}
```

5.3 Dplyr

```
df_dplyr_5 <- function(Posts, Comments){  
  
  # wybieram interesujące nas kolumny  
  CmtTotScr <- Comments %>% select(PostId, UserId, Score)
```

```

# grupuję i liczę sumy Score w grupach
CmtTotScr <- CmtTotScr %>% group_by(PostId, UserId) %>%
  summarise(CommentsTotalScore = sum(Score))

colnames(CmtTotScr)[3] <- "CommentsTotalScore"

# wybieram pytania
Posts1 <- Posts %>% filter(PostTypeId == 1)

# łączę
CmtTotScr <- left_join(Posts1, CmtTotScr, by = c("Id" = "PostId" , "OwnerUserId"="UserId") )

# sortuję u wybieram 10 pierwszych
df1 <- CmtTotScr %>% arrange(desc(CommentsTotalScore)) %>%
  top_n(10) %>% select(Title, CommentsTotalScore)

return(df1)
}

```

5.4 Data.table

```

df_table_5 <- function(Posts, Comments){

# przekształcam na data.table
Posts_table <- as.data.table(Posts)
Comments_table <- as.data.table(Comments)

# sumuję Score po grupach
CmtTotScr <- Comments_table[,.(CommentsTotalScore = sum(Score)),
  by = .(PostId, UserId)]

# sortuję do joina
setkey(CmtTotScr, PostId, UserId)
setkey(Posts_table, Id, OwnerUserId)

# inner join
Posts_table <- Posts_table[CmtTotScr, nomatch = 0 ]

# wybieram pytania
Posts_table <- Posts_table[PostTypeId == 1 ,,]

# wybieram interesujące nas kolumny
Posts_table <- Posts_table[, .(Title, CommentsTotalScore)]

# sortuję i wybieram 10 pierwszych
setorder(Posts_table, -CommentsTotalScore)
dt1 <- Posts_table[1:10,,]
return(dt1)
}

```

5.5 Testy i benchmarki

```
dplyr::all_equal(df_sql_5(Posts, Comments), df_base_5(Posts, Comments))

## [1] TRUE

dplyr::all_equal(df_sql_5(Posts, Comments), df_dplyr_5(Posts, Comments))

## [1] TRUE

dplyr::all_equal(df_sql_5(Posts, Comments), df_table_5(Posts, Comments))

## [1] TRUE

microbenchmark::microbenchmark(
  times = 10,
  sqldf_5 = df_sql_5(Posts, Comments),
  base_5 = df_base_5(Posts, Comments),
  dplyr_5 = df_dplyr_5(Posts, Comments),
  data.table_5 = df_table_5(Posts, Comments))

## Unit: milliseconds
##      expr      min       lq      mean     median        uq
##  sqldf_5  672.5714  705.8861  710.71570  711.11520  716.57886
##  base_5   2043.2687 2073.6650 2136.90561 2121.78358 2168.04385
##  dplyr_5   223.1902  246.2410  262.73887  253.20030  280.48608
## data.table_5  31.5217  33.0953   57.18135   62.91049   70.68224
##      max neval
##  739.50951    10
## 2286.21586    10
##  316.46830    10
##   79.25235    10
```

6 Zadanie 6

6.1 Zapytanie SQL

Wyberzmy tych użytkowników i informacje o nich, którzy mają złotą odznakę, a identyczne odznaki jak oni, ma jedynie od 2 do 10 użytkowników.

```
df_sql_6 <- function(Badges, Users){
  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") -> df1
return(df1)
}

```

6.2 Bazowy R

```

df_base_6 <- function(Badges, Users) {
#Najpierw zliczamy liczbę pojawień się zmiennej "Name", przy "Class" równym 1.
#Następnie zmieniamy nazwy kolumn i wybieramy te, rzędy, w
#których "Count" wynosi między 2 a 10

x <- as.data.frame(table(Badges[Badges$Class == 1, "Name"]), stringsAsFactors = FALSE)
colnames(x) <- c("Name" , "Count")
x <- x[x$Count <= 10 & x$Count >= 2 ,]
rownames(x) <- NULL

#Następnie wybieramy te Nazwy z y , które występują również w x.
y <- Badges[Badges$Class == 1, c("Name", "UserId")]
y <- y[y$Name %in% x$Name,]
rownames(y) <- NULL
ValuableBadges <- y

#Teraz wybieramy z Users te ID, które pojawiają się również w ValuableBadges.
#Następnie sortujemy zapytanie sqlowe, aby było takie same
#jak to, które dostaliśmy w bazowym R. Na koniec sprawdzamy

u <-Users[Users$Id %in% ValuableBadges$UserId, c("Id","DisplayName",
                                                "Reputation","Age","Location")]

rownames(u) <- NULL
u <- u[order(u$Id),]

return(u)
}

```

6.3 Dplyr

```
df_dplyr_6 <- function(Badges, Users) {

#Na początku dsiewam te obserwacje z Badges, które nie mają Class równej 1,
# następnie wybieram nazwę i zliczam jej wystąpienia
x <- Badges %>% filter(Class == 1) %>% select(Name) %>% count(Name)

#zmieniam nazwę 2 kolumny na count
colnames(x)[2] <- "count"

#ponownie filtruję te, które nie mają liczności między 2 a 10
x <- x %>% filter(count >= 2 & count <= 10)

#Biorę te obserwacje z Badges, których nazwa występuje w x,
#następnie biorę te z Class równym jeden i wybieram kolumny Name i UserId.
y <- Badges %>% filter(Badges$Name %in% x$Name) %>%
  filter(Class == 1)%>% select(Name , UserId)

#Wybieram kolumny do Joina.
u <- Users %>% select(Id, DisplayName, Reputation, Age, Location)

#Łączę po Id i UserId.
df1 <- inner_join(u, y, by = c("Id" = "UserId" ))

#Wybieram unikatowe obserwacje.
df1 <- df1 %>% distinct(Id, DisplayName, Reputation, Age, Location)
return(df1)
}
```

6.4 Data.table

```
df_table_6 <- function(Badges, Users) {

#Na początek zamieniam wynik zapytania i potrzebne ramki danych na data.table
Badges_table <- data.table(Badges)
User_table <- data.table(Users)

#Wybieram obserwacje z Class równym 1, następnie grupuję po nazwie i zliczam
#poszczególne wystąpienia nazw jako Count.
x <- Badges_table[Class == 1, .(Count = .N), by = Name]

#Wybieram te obserwacje z Count pomiędzy 2 a 10.
x <- x[Count <=10 & Count >= 2,,]

#Wybieram obserwacje, których nazwa jest w x, mają Class równy 1, oraz zwracam
#kolumny Name i UserId jako data.table
ValuableBadges <- Badges_table[Badges$Name %in% x$Name & Class == 1, .(Name, UserId)]

#Sortuję tablice danych przygotowując je do złączenia. Dzięki setkey markuję je jako posortowane.
setkey(ValuableBadges, UserId)
setkey(User_table, Id)
```

```

#Łączę tablice danych
User_table <- User_table[ValuableBadges, nomatch= 0]

#Wybieram interesujące mnie kolumny, następnie biorę te, których obserwacje są unikatowe
dt1 <- unique(User_table[,.(Id, DisplayName,Reputation, Age, Location),])

#Sortuję tablice, markuję je jako posortowane.
setkey(dt1, Id)
return(dt1)
}

```

6.5 Testy i benchmarki

```

dplyr::all_equal(df_sql_6(Badges, Users), df_base_6(Badges, Users))

## [1] TRUE

dplyr::all_equal(df_sql_6(Badges, Users), df_dplyr_6(Badges, Users))

## [1] TRUE

dplyr::all_equal(df_sql_6(Badges, Users), df_table_6(Badges, Users))

## [1] TRUE

microbenchmark::microbenchmark(
  times = 10,
  sqldf_6 = df_sql_6(Badges, Users),
  base_6 = df_base_6(Badges, Users),
  dplyr_6 = df_dplyr_6(Badges, Users),
  data.table_6 = df_table_6(Badges, Users))

## Unit: milliseconds
##      expr      min       lq      mean     median      uq
##  sqldf_6 365.831092 367.073361 379.544831 378.705291 391.285459
##   base_6   3.019846   3.125425   6.179531   3.567276   4.228694
##  dplyr_6  17.532135  17.815368  21.499767  18.795759  23.347356
## data.table_6 14.612327 14.947617  20.967742  17.670151  23.317431
##      max neval
## 396.22776    10
##  29.58380    10
##  34.56273    10
##  43.96792    10

```

7 Zadanie 7

7.1 Zapytanie SQL

Wyberzmy 10 pytań które dostały najwięcej pozytywnych głosów, stworzonych przed 2016 rokiem, które nie dostały żadnej odpowiedzi po 2016 roku.

```

df_sql_7 <- function(Posts, Votes){
  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
  ") -> df1
  return(df1)
}

```

7.2 Base R

```

df_base_7 <- function(Posts, Votes) {

  # wybieram pozytywne głosy
  V <- as.data.frame(Votes[Votes$VoteTypeId ==2, "PostId"], stringsAsFactors = FALSE)
  colnames(V)[1] <- "PostId"

  # wybieram rok
  V$VoteDate <- as.integer( substring(Votes[Votes$VoteTypeId ==2, "CreationDate"] , 1,4))

  # lata 2016 i wyżej nazywam new, resztę old (ifelse jest bardzo szybką funkcją)
  V$VoteDate <- ifelse(V$VoteDate >= 2016, "new", "old")

  # liczę wystąpienia w grupach

```

```

VotesByAge <- aggregate(V$PostId, by = V[, c("PostId", "VoteDate")], length)
colnames(VotesByAge)[3] <- "Total"

V <- VotesByAge

# jeżeli new, to daję odpowiadający mu total, jak nie to 0
V$NewVotes <- ifelse(V$VoteDate == "new", V$Total, 0)
# analogicznie
V$OldVotes <- ifelse(V$VoteDate == "old", V$Total, 0)

# tutaj będziemy musieli kilkakrotnie zliczać i łączyć
# liczę maksima dla danego postId
Vnew <- aggregate(V$NewVotes, by = list(V$PostId), max )
Vold <- aggregate(V$OldVotes, by = list(V$PostId), max )
# a następnie sumę
Vtot <- aggregate(V$Total, by = list(V$PostId), sum)

# teraz łączę poprzednie wyniki w całość
Vall <- merge(Vnew, Vold, "Group.1")
Vall <- merge(Vall, Vtot, "Group.1")

# porządkuję
colnames(Vall) <- c("PostId", "NewVotes", "OldVotes", "Votes")

# wybieram te, które nie mają nowych głosów
VotesByAge2 <- Vall[Vall$NewVotes == 0,]

# wybieram pytania
P <- Posts[Posts$PostTypeId == 1,]

# łączę
A <- merge(VotesByAge2, P, by.x = "PostId", by.y = "Id")

# porządkuję
A <- A[, c("Title", "OldVotes")]
A <- A[order(-A$OldVotes),]
A$OldVotes <- as.integer(A$OldVotes)
rownames(A) <- NULL

final <- head(A, n=10)
return(final)
}

```

7.3 Dplyr

```

df_dplyr_7 <- function(Posts, Votes){

# korzystając z szybkiej funkcji year ekstraktuję rok z daty
VotesByAge <- Votes %>% mutate( CreationDate = year(CreationDate)) %>%
  # następnie przypisuję new do lat 2016 i wyższych a old do reszty

```

```

mutate(VoteDate = if_else(CreationDate == 2016 |
                          CreationDate == 2017, 'new', 'old' )) %>%

# wybieram pozytywne głosy
filter( VoteTypeId == 2) %>%
# grupuję i zliczam
group_by(PostId, VoteDate) %>%
count() %>% rename( Total = n )

```

```

VotesByAge2 <- VotesByAge %>% group_by(PostId) %>%
  # tworzę kolumny z wartościami maksymalną głosów zarówno dla new jak i old.
  # np. dla new zwracam wektor Total, z którego biorę maksymalną wartość dla
  # odpowiedniego PostId. Jeśli mam do czynienia z old, zwracam zero, by nie zaburzać
  # wyników
  mutate(NewVotes = max(if_else(VoteDate == 'new', Total, as.integer(0)))) %>%
  mutate(OldVotes = max(if_else(VoteDate == 'old', Total, as.integer(0)))) %>%
  filter(NewVotes == 0)
  # pomijam sql-owe Sum(total) jako Votes, ponieważ nic z nim później nie robimy.

# łączę z Posts by wybrać tytuł
A <- inner_join(VotesByAge2, Posts, by = c("PostId" = "Id")) %>% filter(PostTypeId == 1) %>%
  ungroup() %>% select(Title, OldVotes) %>%
  # porządkuję
  arrange(desc(OldVotes)) %>% as.data.frame() %>%
  slice(1:10)

return(A)
}

```

7.4 Data.table

```

df_table_7 <- function(Posts, Votes) {

# zmieniam na data.table
Ptable <- as.data.table(Posts)
Vtable <- as.data.table(Votes)

# wybieram pozytywne głosy
A <- Vtable[Vtable$VoteTypeId ==2 ,,]

# wybieram rok z daty
A$VoteDate <- A[, .(VoteDate = year(A$CreationDate)),]

# dla 2016 i wyżej przypisuję "new", dla reszty "old"
A$VoteDate <- A[, .(VoteDate= ifelse(VoteDate >= 2016 , "new", "old")) ,]

# zliczam wystąpienia w grupach
VotesByAge <- A[, .(Total = .N) , by = .(PostId, VoteDate)]

# liczę maksima w grupach i sumę jako Total
VotesByAge2 <- VotesByAge[, .(NewVotes = max(ifelse(VoteDate == "new", Total, as.integer(0))),
                             OldVotes = max(ifelse(VoteDate == "old", Total, as.integer(0))),

```

```

# wybieram te, które nie mają nowych głosów
Votes = sum(Total), by = .(PostId)][NewVotes ==0,,]

# sortuję do złączenia
setkey(VotesByAge2, PostId)
setkey(Ptable, Id)

# inner join
V <- VotesByAge2[Ptable, nomatch =0]

# wybieram pytania
V <- V[PostTypeId ==1, .(Title, OldVotes)]

# sortuję i wybieram 10 pierwszych wyników
V <- setorder(V, -OldVotes)
V <- V[1:10]
return(V)
}

```

7.5 Testy i benchmarki

```

dplyr::all_equal(df_sql_7(Posts, Votes), df_base_7(Posts, Votes))

## [1] TRUE

dplyr::all_equal(df_sql_7(Posts, Votes), df_dplyr_7(Posts, Votes))

## [1] TRUE

dplyr::all_equal(df_sql_7(Posts, Votes), df_table_7(Posts, Votes))

## [1] TRUE

microbenchmark::microbenchmark(
  times = 10,
  sqldf_7 = df_sql_7(Posts, Votes),
  base_7 = df_base_7(Posts, Votes),
  dplyr_7 = df_dplyr_7(Posts, Votes),
  data.table_7 = df_table_7(Posts, Votes))

## Unit: seconds
##      expr      min       lq      mean     median        uq       max
##  sqldf_7  1.189787  1.196639  1.224618  1.201948  1.234400  1.321646
##   base_7  3.129279  3.246158  3.356017  3.341634  3.475629  3.617847
##  dplyr_7 18.015265 18.490462 18.673001 18.555040 19.085396 19.635313
## data.table_7 3.721704 3.918021 3.984381 3.998488 4.092979 4.121722
## neval
##      10
##      10
##      10
##      10

```