PDU 2023/2024

Praca domowa (praca domowa nr 1) (max. = 40 p.)

Maksymalna ocena: 40 p.

Prace domowe należy przesłać za pośrednictwem platformy `Moodle` – **jeden plik .R** o nazwie `Imie_Nazwisko_nrAlbumu_PD1.R`.

Plik powinien zawierać rozwiązanie zadań zgodne z załączonym szablonem. Uwaga: nazwy plików nie powinny zawierać polskich liter diakrytyzowanych (przekształć $ą \to a$ itd.).

1 Zbiory danych

Będziemy pracować na uproszczonym zrzucie zanonimizowanych danych z serwisu https://travel.stackexchange.com/ (na marginesie: pełen zbiór danych dostępny jest pod adresem https://archive.org/details/stackexchange), który składa się z następujących ramek danych:

- Posts.csv.gz
- Users.csv.gz
- Comments.csv.gz
- PostLinks.csv.gz
- Votes.csv.gz

Przed przystąpieniem do rozwiązywania zadań zapoznaj się z ww. serwisem oraz znaczeniem poszczególnych kolumn we wspomnianych ramkach danych, zob. https://ia600107.us.archive.org/27/items/stackexchange/readme.txt

Przykładowe wywołanie — ładowanie zbioru `Posts`:

```
# ww. pliki pobralismy do katalogu travel_stackexchange_com/
Posts <- read.csv("travel_stackexchange_com/Posts.csv.gz")
head(Posts)
```

Uwaga: Nazwy ramek danych po wczytaniu zbiorów powinny wyglądać następująco: `Posts`, `Users`, `Votes`, `Comments` oraz `PostLinks`.

2 Informacje ogólne

Rozwiąż poniższe zadania przy użyciu wywołań funkcji bazowych oraz tych, które udostępniają pakiety `dplyr` oraz `data.table` – nauczysz się ich samodzielnie; ich dokumentację znajdziesz łatwo w internecie. Każdemu z **5** poleceń SQL powinny odpowiadać cztery równoważne sposoby ich implementacji w R, kolejno:

1. `sqldf::sqldf()`;
2. tylko funkcje bazowe;
3. `dplyr`;
4. `data.table`.

Rozwiązanie każdego zadania powinno być zaimplementowane jako funkcje: `sql_i()`, `base_i()`, `dplyr_i()`, `table_i()`, o nazwach i parametrach określonych w szablonie.

W przypadku każdego zadania:

1. Upewnij się, że zwracane wyniki są ze sobą tożsame (ewentualnie z dokładnością do permutacji wierszy wynikowych ramek danych, zob. np. funkcję `dplyr::all_equal` lub `compare::compare`).
2. Kod rozwiązań opatrz komentarzami oraz podaj słowną interpretację (tzn. intuicyjne – „dla laika" – tłumaczenie) każdego zapytania.
3. W każdym przypadku porównaj czasy wykonania napisanych przez Ciebie wyrażeń przy użyciu jednego wywołania `microbenchmark::microbenchmark()`, np.:

```
microbenchmark::microbenchmark(
   sqldf = sql_i(...),
   base = base_i(...),
   dplyr = dplyr_i(...),
   data.table = table_i(...)
```

UWAGA

Wysyłając rozwiązanie upewnij się, że plik jest zgodny z szablonem rozwiązania, tzn. nazwy funkcji i ich parametrów nie zostały zmienione oraz wskazane fragmenty kodu zostały zakomentowane.

3 Zadania do rozwiązania

```
--- 1)
SELECT STRFTIME('%Y', CreationDate) AS Year,
       STRFTIME('%m', CreationDate) AS Month,
       COUNT(*) AS TotalAccountsCount,
       AVG(Reputation) AS AverageReputation
FROM Users
GROUP BY Year, Month
```

```
--- 2)
SELECT Users.DisplayName, Users.Location, Users.Reputation,
       STRFTIME('%Y-%m-%d', Users.CreationDate) AS CreationDate,
       Answers.TotalCommentCount
FROM (
        SELECT OwnerUserId, SUM(CommentCount) AS TotalCommentCount
        FROM Posts
        WHERE PostTypeId == 2 AND OwnerUserId != ''
        GROUP BY OwnerUserId
     ) AS Answers
JOIN Users ON Users.Id == Answers.OwnerUserId
ORDER BY TotalCommentCount DESC
LIMIT 10
```

```sql
--- 3)
SELECT Spam.PostId, UsersPosts.PostTypeId, UsersPosts.Score,
       UsersPosts.OwnerUserId, UsersPosts.DisplayName,
       UsersPosts.Reputation
FROM (
        SELECT PostId
        FROM Votes
        WHERE VoteTypeId == 12
     ) AS Spam
JOIN (
        SELECT Posts.Id, Posts.OwnerUserId, Users.DisplayName,
               Users.Reputation, Posts.PostTypeId, Posts.Score
        FROM Posts JOIN Users
        ON Posts.OwnerUserId = Users.Id
     ) AS UsersPosts
ON Spam.PostId = UsersPosts.Id
```

```sql
--- 4)
SELECT Users.Id, Users.DisplayName, Users.UpVotes, Users.DownVotes, Users.Reputation,
       COUNT(*) AS DuplicatedQuestionsCount
FROM (
        SELECT Duplicated.RelatedPostId, Posts.OwnerUserId
        FROM (
                SELECT PostLinks.RelatedPostId
                FROM PostLinks
                WHERE PostLinks.LinkTypeId == 3
             ) AS Duplicated
        JOIN Posts
        ON Duplicated.RelatedPostId = Posts.Id
     ) AS DuplicatedPosts
JOIN Users ON Users.Id == DuplicatedPosts.OwnerUserId
GROUP BY Users.Id
HAVING DuplicatedQuestionsCount > 100
ORDER BY DuplicatedQuestionsCount DESC
```

```sql
--- 5)
SELECT QuestionsAnswers.Id,
       QuestionsAnswers.Title,
       QuestionsAnswers.Score,
       MAX(Duplicated.Score) AS MaxScoreDuplicated,
       COUNT(*) AS DulicatesCount,
       CASE
         WHEN QuestionsAnswers.Hour < '06' THEN 'Night'
         WHEN QuestionsAnswers.Hour < '12' THEN 'Morning'
         WHEN QuestionsAnswers.Hour < '18' THEN 'Day'
         ELSE 'Evening'
         END DayTime
FROM (
        SELECT Id, Title,
               STRFTIME('%H', CreationDate) AS Hour, Score
        FROM Posts
        WHERE Posts.PostTypeId IN (1, 2)
     ) AS QuestionsAnswers
JOIN (
        SELECT PL3.RelatedPostId, Posts.Score
        FROM (
               SELECT RelatedPostId, PostId
               FROM PostLinks
               WHERE LinkTypeId == 3
             ) AS PL3
        JOIN Posts ON PL3.PostId = Posts.Id
     ) AS Duplicated
ON QuestionsAnswers.Id = Duplicated.RelatedPostId
GROUP BY QuestionsAnswers.Id
ORDER By DulicatesCount DESC
```