PDU 2021/2022

Praca domowa nr 4 (max. = 25 p.)

Prace domowe należy przesłać za pośrednictwem platformy `Moodle` – **jedno archiwum `.zip`**[1] o nazwie typu `Nazwisko_Imie_PD4.zip`. W archiwum znajdować się powinien jeden katalog, `Nazwisko_Imie_PD4`, dopiero w którym umieszczone zostaną następujące pliki:

- `Nazwisko_Imie_PD4.ipynb` (rozwiązanie zadań)
- `Nazwisko_Imie_PD4.html` (ściągnięta wersja powyższego w formacie `html` - zob. `File -> Downlowad as -> html` w notatniku `Jupyter`).

  Nazwy plików nie powinny zawierać polskich liter diakrytyzowanych (przekształć $ą \rightarrow a$ 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:

- Badges.csv.gz
- Comments.csv.gz
- PostLinks.csv.gz
- Posts.csv.gz
- Tags.csv.gz
- Users.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 `Tags`:

```
import pandas as pd
import numpy as np

Tags = pd.read_csv("travel_stackexchange_com/Tags.csv.gz",
                   compression = 'gzip')
Tags.head()
```

Każdą z ramek danych należy wyeksportować do bazy danych SQLite przy użyciu wywołania metody `to_sql()` w klasie `pandas.DataFrame`. Dokładniej, pracę z bazą danych możemy pzeprowadzić w następujący sposób.

```
import os, os.path
import sqlite3

# sciezka dostępu do bazy danych:
baza = os.path.join('przyklad.db')
```

[1] A więc nie: .rar, .7z itp.

```
if os.path.isfile(baza): # jesli baza już istneje...
    os.remove(baza)      # ...usuniemy ja
                         # zeby miec zaczac z ,,czystą kartą''

conn = sqlite3.connect(baza)      # połączenie do bazy danych

Badges.to_sql("Badges", conn)     # importujemy ramkę danych do bazy danych
Comments.to_sql("Comments", conn)
PostLinks.to_sql("PostLinks", conn)
Posts.to_sql("Posts", conn)
Tags.to_sql("Tags", conn)
Users.to_sql("Users", conn)
Votes.to_sql("Votes", conn)

#
pd.read_sql_query("""
                  Zaptanie SQL
                  """, conn)
# ...
# rozwiązania zadan
# po skończonej pracy zamykamy połączenie
#
conn.close()
```

W szczególności należy zagwarantować, że w każdym przypadku wynik jest klasy `DataFrame` a nie `Series`.

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

2 Informacje ogólne

Rozwiąż poniższe zadania przy użyciu wywołań funkcji i metod z pakietu `pandas`. Każdemu z **5** poleceń SQL powinny odpowiadać dwa równoważne sposoby ich implementacji, kolejno:

1. wywołanie `pandas.read_sql_query("""zapytanie SQL""")`;
2. wywołanie ciągu „zwykłych'' metod i funkcji z pakietu `pandas`.

Upewnij się, że zwracane wyniki są ze sobą tożsame (ewentualnie z dokładnością do permutacji wierszy wynikowych ramek danych), por. np. metodę `.equals()` z pakietu `pandas`.

Wszystkie rozwiązania umieść w jednym (estetycznie sformatowanym) raporcie `Jupyter`. Za bogate komentarze do kodu i dyskusję max. 5 p.

3 Zadania do rozwiązania

```
--- 1)
SELECT Count, TagName
FROM Tags
WHERE Count > 1000
ORDER BY Count DESC
```

```sql
--- 2)
SELECT Location, COUNT(*) AS Count
FROM (
    SELECT Posts.OwnerUserId, Users.Id, Users.Location
    FROM Users
    JOIN Posts ON Users.Id = Posts.OwnerUserId
)
WHERE Location NOT IN ('')
GROUP BY Location
ORDER BY Count DESC
LIMIT 10
```

```sql
--- 3)
SELECT Year, SUM(Number) AS TotalNumber
FROM (
    SELECT
        Name,
        COUNT(*) AS Number,
        STRFTIME('%Y', Badges.Date) AS Year
    FROM Badges
    WHERE Class = 1
    GROUP BY Name, Year
)
GROUP BY Year
ORDER BY TotalNumber
```

```sql
--- 4)
SELECT
    Users.AccountId,
    Users.DisplayName,
    Users.Location,
    AVG(PostAuth.AnswersCount) as AverageAnswersCount
FROM
(
    SELECT
        AnsCount.AnswersCount,
        Posts.Id,
        Posts.OwnerUserId
    FROM (
            SELECT Posts.ParentId, COUNT(*) AS AnswersCount
            FROM Posts
            WHERE Posts.PostTypeId = 2
            GROUP BY Posts.ParentId
          ) AS AnsCount
    JOIN Posts ON Posts.Id = AnsCount.ParentId
) AS PostAuth
JOIN Users ON Users.AccountId=PostAuth.OwnerUserId
GROUP BY OwnerUserId
ORDER BY AverageAnswersCount DESC, AccountId ASC
LIMIT 10
```

```sql
--- 5)
SELECT Posts.Title, Posts.Id,
       STRFTIME('%Y-%m-%d', Posts.CreationDate) AS Date,
       VotesByAge.Votes
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 '2021' THEN 'new'
                    WHEN '2020' THEN 'new'
                    ELSE 'old'
                    END VoteDate,
                COUNT(*) AS Total
            FROM Votes
            WHERE VoteTypeId IN (1, 2, 5)
            GROUP BY PostId, VoteDate
        ) AS VotesDates
        GROUP BY VotesDates.PostId
        HAVING NewVotes > OldVotes
) AS VotesByAge ON Posts.Id = VotesByAge.PostId
WHERE Title NOT IN ('')
ORDER BY Votes DESC
LIMIT 10
```