Meno: Zaťovič Dominik (ID: 121058)
Predmet: Databázové systémy
Ročník: LS 2023/2024

ZADANIE 2

Popis zadania:

Zadanie je zamerané na vytvorenie základných SQL dotazov nad priloženou PostgreSQL databázou, ktorá vznikla z Stack Exchange Data Dump Superuser datasetu. Cieľom je realizovať nižšie uvedené úlohy ako RESTful endpointy, ktoré sú realizované ako SQL dotazy, transformované do JSON výstupu. Výstup je opísaný ako JSON Schema. Vstupy na pripojenie k databázovému serveru budú poskytované rovnako ako v zadani 1 (pomocou environment premenných).

Poradie vo výstupe musí byť zhodné s jeho definíciou pri jednotlivých end-pointoch. Pri realizácii je možné používať iba čisté SQL dopyty a nie je dovolené používať žiadne ORM.

Okrem implementovania samotných endpointov je potrebné vyhotoviť dokumentáciu, ktorá bude obsahovať:

SQL dopyty s ich popisom, príklady volania HTTP end-pointu (pre každý endpoint).

Začiatok end-pointu:

```python
from fastapi import APIRouter, FastAPI, HTTPException, Query
import psycopg2
import os
from dotenv import load_dotenv

router = APIRouter()

def connect_to_postgres():
    try:
        # Connect to PostgreSQL database
        load_dotenv()
        db_name=os.getenv('DATABASE_NAME')
        db_user=os.getenv('DATABASE_USER')
        db_pass=os.getenv('DATABASE_PASSWORD')
        db_host=os.getenv('DATABASE_HOST')
        db_port=os.getenv('DATABASE_PORT')

        conn = psycopg2.connect(
            database=db_name,
            user=db_user,
            password=db_pass,
            host=db_host,
            port=db_port
        )
        return conn
    except psycopg2.Error as e:
        raise HTTPException(status_code=500, detail=str(e))
```


FIIT STU
Ilkovičova 8, Bratislava

V tejto funkcii sa pripojím na databázu, a zadám do nej informácie potrebné k jej pripojeniu.

Tieto informácie mám uložené v súbore „.env“.

HTTP Volania

GET /v2/posts/:post id/users

```python
def get_post_comments(post_id):
    try:
        conn = connect_to_postgres()
        cur = conn.cursor()

        # Execute query to get users who commented on the specified post, sorted by creation time
        cur.execute("""
            SELECT users.id, users.reputation, users.creationdate, users.displayname, users.lastaccessdate,
                   users.websiteurl, users.location, users.aboutme, users.views, users.upvotes, users.downvotes,
                   users.profileimageurl, users.age, users.accountid
            FROM users
            JOIN comments ON users.id = comments.userid
            WHERE comments.postid = %s
            ORDER BY comments.creationdate DESC;
        """, (post_id,))
        users_data = cur.fetchall()

        # Close cursor and connection
        cur.close()
        conn.close()

        return users_data
    except psycopg2.Error as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Vo funkcii get_post_comments pomocou SQL query vyberieme pomocou SELECT všetky potrebné data z kolónky „users“ spoločne s komentárom daného používateľa, pre ktoré platí že ID užívateľa je rovnaké ako ID komentáru (priradíme komentár k používateľovi) a zároveň rovnaké ako ID postu (ID postu vstupuje do funkcie ako premenna, ktorú voláme cez http). To usporiadame podľa času vytvorenia tohto komentára od najnovšieho po najstaršie.

Táto funkcia nám teda vráti list všetkých používateľov, ktorý komentovali na príspevok s ID, ktoré zadáme, pričom v tomto liste sa pre každého usera nachádzaju aj dodatočné informácie z query pri „SELECT“. Tieto informácie ďalej dáme do printu vo funkcii nižšie.

```python
@router.get("/v2/posts/{post_id}/users")
async def get_post_users(post_id: int):
    users_data = get_post_comments(post_id)
    if not users_data:
        raise HTTPException(status_code=404, detail="No users found for the specified post.")
    else:
        formatted_users = [{
            "id": user[0],
            "reputation": user[1],
            "creationdate": user[2],
            "displayname": user[3],
            "lastaccessdate": user[4],
            "websiteurl": user[5],
            "location": user[6],
            "aboutme": user[7],
            "views": user[8],
            "upvotes": user[9],
            "downvotes": user[10],
            "profileimageurl": user[11],
            "age": user[12],
            "accountid": user[13]
        } for user in users_data]

        formatted_response = {"items": formatted_users}
        return formatted_response
```

Toto je hlavná funkcia pre HTTP volanie. Na začiatku zavoláme funkciu get_post_comments, ktorú som opísal vyššie, ktorá nám vráti list s informáciami o užívateľoch, ktorý komentovali na daný príspevok, zoradený od najnovšieho po najstarší.

Tých potom vo for-loope pridám do dictonary formatted_users a vypíšem potrebné informácie pomocou indexov v liste.

Príklad:

Listing 1: GET /v2/posts/1819157/users


```
{
  "items": [
    {
      "id": 1866388,
      "reputation": 1,
      "creationdate": "2023-12-01T00:05:24.337000+01:00",
      "displayname": "TomR.",
      "lastaccessdate": "2023-12-03T06:18:19.607000+01:00",
      "websiteurl": null,
      "location": null,
      "aboutme": null,
      "views": 1,
      "upvotes": 0,
      "downvotes": 0,
      "profileimageurl": null,
      "age": null,
      "accountid": 30035903
    }
  ]
}
```

GET /v2/users/:user id/friends

```python
@router.get("/v2/users/{user_id}/friends")
async def get_discussing_users(user_id: int):
    try:
        conn = connect_to_postgres()
        cur = conn.cursor()

        # Retrieve distinct users who commented on the specified user's posts or on posts the specified user commented
        cur.execute("""
            SELECT DISTINCT u.id, u.reputation, u.creationdate, u.displayname, u.lastaccessdate,
                   u.websiteurl, u.location, u.aboutme, u.views, u.upvotes, u.downvotes,
                   u.profileimageurl, u.age, u.accountid
            FROM users u
            JOIN comments c ON u.id = c.userid
            JOIN posts p ON c.postid = p.id
            WHERE p.owneruserid = %s OR c.userid = %s
            ORDER BY u.creationdate;
        """, (user_id, user_id))

        discussing_users_data = cur.fetchall()

        # Close cursor and connection
        cur.close()
        conn.close()

        # Construct the formatted response
        formatted_users = [{
```

V tejto funkcii zavolám SQL query príkaz, ktorý mi vyberie potrebné informácie o používateľovi z kolónky „users“ spolu s ich komentárom, (ID užívateľa je rovnaké ako ID komentáru), a k ním aj príspevok (ID príspevku je rovnaké ako ID komentáru).

Toto bude platiť len vtedy, ak majiteľ príspevku je rovnaký ako ID nami zadané, alebo ID užívateľa, ktorému patrí komentár pod iným príspevkom (nie je jeho majiteľ) je rovnaké ako ID nami zadané. To zoradíme podľa dátumu vytvorenia účtu používateľa.

Tak ako v predošlej funkcii, toto SQL mi vráti list užívateľov s ich informáciami, ktorý následne rovnakým spôsobom ako v predošlej funkcii dám do dictionary.

Príklad:

Listing 2: GET /v2/users/1076348/users


```
{
  "items": [
    {
      "id": 482362,
      "reputation": 10581,
      "creationdate": "2015-08-11T17:42:36.267000+02:00",
      "displayname": "DrZoo",
      "lastaccessdate": "2023-12-03T06:41:11.750000+01:00",
      "websiteurl": null,
      "location": null,
      "aboutme": null,
      "views": 1442,
      "upvotes": 555,
      "downvotes": 46,
      "profileimageurl": null,
      "age": null,
      "accountid": 2968677
    },
    {
      "id": 1076348,
      "reputation": 1,
      "creationdate": "2019-08-15T16:00:28.473000+02:00",
      "displayname": "Richard",
      "lastaccessdate": "2019-09-10T16:57:48.527000+02:00",
      "websiteurl": null,
      "location": null,
      "aboutme": null,
      "views": 0,
      "upvotes": 0,
      "downvotes": 0,
      "profileimageurl": null,
      "age": null,
      "accountid": 16514661
    }
  ]
}
```

GET /v2/tags/:tagname/stats

```python
@router.get("/v2/tags/{tagname}/stats")
async def get_tag_stats(tagname: str):
    try:
        conn = connect_to_postgres()
        cur = conn.cursor()

        # SQL query to retrieve count of posts with the specified tag for each day of the week
        cur.execute("""
            SELECT EXTRACT(DOW FROM p.creationdate) AS day_of_week, COUNT(p.id)
            FROM posts p
            JOIN post_tags pt ON p.id = pt.post_id
            JOIN tags t ON pt.tag_id = t.id
            WHERE t.tagname = %s
            GROUP BY day_of_week
            ORDER BY day_of_week;
        """, (tagname,))
        tag_posts_count = cur.fetchall()


        # SQL query to retrieve total count of posts published on each day of the week
        cur.execute("""
            SELECT EXTRACT(DOW FROM creationdate) AS day_of_week, COUNT(id)
            FROM posts
            GROUP BY day_of_week
            ORDER BY day_of_week;
        """)
        total_posts_count = cur.fetchall()
```

V tejto funkcii zavolám SQL query kód, ktorý mi vyberie dni v týždni 0- nedeľa, 1- pondelok...) a počet ID príspevkov. Následne spojím tabuľky posts a post_tags na základe ich ID a tabuľky tags a post_tags na základe ich ID. To jest, ku každému postu zistím ich post tag a tag. Následne vyfiltrujem len tie, ktoré majú rovnaký tagname ako námi zadaný. Groupnem ich na základe toho, aký je to deň a zoradím ich. Týmto efektívne zistím, koľko postov je pre každý deň pre daný tag.

V druhom SQL kóde zistím počet všetkých postov pre daný deň (je jedno aký tagname).

```
    # Remap the days of the week to match your database's representation
    day_mapping = {
        1: "monday",
        2: "tuesday",
        3: "wednesday",
        4: "thursday",
        5: "friday",
        6: "saturday",
        0: "sunday"
    }

    # Combine the results into a dictionary with percentage representation
    tag_stats = {}
    for tag_count, total_count in zip(tag_posts_count, total_posts_count):
        day_of_week = int(tag_count[0])  # Extract day of the week
        tag_percentage = round(tag_count[1] / total_count[1] * 100, 2) if total_count[1] > 0 else 0
        # Map day of the week to its corresponding percentage
        tag_stats[day_mapping[day_of_week]] = tag_percentage

    # Close cursor and connection
    cur.close()
    conn.close()

    return {"result": tag_stats}

except psycopg2.Error as e:
    raise HTTPException(status_code=500, detail=str(e))
```

Následne vo for-loope pre každý deň v týždni zistím percentuálne zastúpenie príspevok s našim tagov oproti všetkým tagom a výsledok pridám do dictionary na im prislúchajúcu pozíciu.

Priklad:

Listing 3: GET /v2/tags/linux/stats


```
{
  "result": {
      "sunday": 4.88,
      "monday": 4.71,
      "tuesday": 4.69,
      "wednesday": 4.63,
      "thursday": 4.57,
      "friday": 4.67,
      "saturday": 4.98
  }
}
```

GET /v2/posts/?duration=:duration in minutes&limit=:limit

GET /v2/posts?limit=:limit&query=:query

```python
@router.get("/v2/posts/")
async def get_recent_posts(duration: int = Query(None), limit: int = Query(...), query: str = Query(None)):
    try:
        conn = connect_to_postgres()
        cur = conn.cursor()
        if duration:
        # SQL query to retrieve the `limit` newest resolved posts opened within the last `duration` minutes
            cur.execute("""
                SELECT id, creationdate, viewcount, lasteditdate, lastactivitydate, title, closeddate,
                ROUND(EXTRACT(EPOCH FROM (closeddate - creationdate)) / 60::numeric, 2) AS duration
                FROM posts
                WHERE closeddate IS NOT NULL
                AND EXTRACT(EPOCH FROM (closeddate - creationdate)) / 60 <= %s
                ORDER BY creationdate DESC
                LIMIT %s;
            """, (duration, limit))
            recent_posts = cur.fetchall()

            # Close cursor and connection
            cur.close()
            conn.close()

            # Format the duration of opening for each post
            formatted_posts = []
```

Keďže úloha 4 a 5 sa spúštajú rovnakým url, musel som ich dať do jednej funkcie. Funkcia číta, či mi do nej vstupuje duration alebo query. To, ktoré nevstúpi, bude None, druhé sa prepíše na hodnotu, akú zadáme.

Následne vykoná 1 z 2 častí mojej funkcie, podľa toho či sme zadali duration alebo query.

Ak mi vstúpi duration (úloha 4):

Zavolám SQL kód, ktorý mi cez SELECT vyberie potrebné informácie a spolu s tým aj dĺžku času, ako boli otvorené (closeddate mínus creationdate) – to mi vráti čas v sekundách, preto treba vydeliť 60 (na minúty) a zaokrúhli to na 2 desatinné miesta.

Toto mi berie z kolónky posts, v prípade že closeddate existuje (nie je NULL) a v prípade že čas otvorenia bol menší ako čas nami zadaný.

Tieto posty následne zoradím podľa dátumu vytvorenia od najnovšieho po najstaršie a vyberiem ich pomocou LIMIT len toľko, koľko sme zadali.

Príklad:

Listing 4: GET /v2/posts?duration=5&limit=2


```
{
  "items": [
    {
      "id": 1818849,
      "creationdate": "2023-11-30T16:55:32.137000+01:00",
      "viewcount": 22924,
      "lasteditdate": null,
      "lastactivitydate": "2023-11-30T16:55:32.137000+01:00",
      "title": "Why is my home router address is 10.x.x.x and not 100.x.x.x which is properly reserved and widely accepted for CGNAT?",
      "closeddate": "2023-11-30T16:59:23.560000+01:00",
      "duration": 3.86
    },
    {
      "id": 1818386,
      "creationdate": "2023-11-27T18:26:57.617000+01:00",
      "viewcount": 19,
      "lasteditdate": null,
      "lastactivitydate": "2023-11-27T18:26:57.617000+01:00",
      "title": "Are there any libraries for parsing DWG files with LGPL, MIT, Apache, BSD?",
      "closeddate": "2023-11-27T18:29:18.947000+01:00",
      "duration": 2.36
    }
  ]
}
```


Ak mi vstúpi do funkcie query (úloha 5):

```
else:
    if query:
        # Search for posts containing the query string in title or body (case-insensitive)
        cur.execute("""
            SELECT p.id, p.creationdate, p.viewcount, p.lasteditdate, p.lastactivitydate, p.title, p.body, p.answercount, p.closeddate, t.tagname
            FROM posts p
            JOIN post_tags pt ON p.id = pt.post_id
            JOIN tags t ON pt.tag_id = t.id
            WHERE LOWER(p.title) LIKE %s OR LOWER(p.body) LIKE %s
            ORDER BY p.creationdate DESC
            LIMIT %s;
        """, ('%' + query.lower() + '%', '%' + query.lower() + '%', limit))
```

Zavolám SQL kód, ktorí mi vyberie potrebné informácie z príspevkov, spojí ich s post_tags na základe totožného ID a taktiež s tags. To spraví pre posty, kde v title alebo body príspevku sa nachádza slovo rovnaké, ako sme zadali na vstupe a zoradíme ich od najnovšieho po najstaršie. Zoberieme toľko postov, koľko ich zadáme v limite.

```
else:
    # Fetch posts without any query filter
    cur.execute("""
        SELECT p.id, p.creationdate, p.viewcount, p.lasteditdate, p.lastactivitydate, p.title, p.body, p.answercount, p.closeddate, t.tagname
        FROM posts p
        LEFT JOIN post_tags pt ON p.id = pt.post_id
        LEFT JOIN tags t ON pt.tag_id = t.id
        ORDER BY p.creationdate DESC
        LIMIT %s;
    """, (limit,))

posts_data = cur.fetchall()
```

Ak užívateľ nezadá žiadne query, tak to vyberie LIMIT postov, zoradených od najnovších, bez filtrovanie akéhokoľvek slova, teda akékoľvek príspevky, ktorých ID je rovnaké ako ID tagu a post_tags.

Vo všetkých prípadoch sa mi vráti list, ktorý následne dám do dictonary.

Príklad:

Listing 5: GET /v2/posts?limit=2&query=linux


```
{
  "items": [
    {
      "id": 1819160,
      "creationdate": "2023-12-03T05:22:43.587000+01:00",
      "viewcount": 7,
      "lasteditdate": null,
      "lastactivitydate": "2023-12-03T05:22:43.587000+01:00",
      "title": "Keyboard not working on khali linux",
      "body": "<p>I have recently installed virtualbox on my windows 10 and trying to run Linux Ubuntu and Kali. Everything working on Ubuntu without any issue bu
      bluetooth 500) input. Please can anyone help me out here.\nMany thanks in advance!!</p>\n",
      "answercount": 0,
      "closeddate": null,
      "tags": "virtual-machine"
    },
    {
      "id": 1819154,
      "creationdate": "2023-12-03T04:24:20.410000+01:00",
      "viewcount": 9,
      "lasteditdate": null,
      "lastactivitydate": "2023-12-03T04:24:20.410000+01:00",
      "title": "Motherboard causes Windows boot into INACCESSIBLE BOOT DEVICE",
      "body": "<p>Alternative problem:<br />\nAfter replacing motherboard(MB) or resetting BIOS settings to defaults, Windows boots into INACCESSIBLE BOOT DEVICE
      mostly about noticeable failures of software or hardware. In this case, no software or hardware malfunction is expected, but BSOD happens.</p>\n<p>Main symp
      PC/MB (or other hardware):</p>\n<ul>\n<li>Boot from SATA works well</li>\n</ul>\n<p>On the new specific PC/MB:</p>\n<ul>\n<li>SATA boot: ends with IBD BSOD<
      OK</li>\n<li>When trying to install W10, SSD or M.2 are not detected (unless drivers are manually provided?)</li>\n<li>Boot with Live USB (Clonezilla or Lin
      correctly</li>\n</ul>\n<p>How to fix W10 boot BSOD?</p>\n",
      "answercount": 1,
      "closeddate": null,
      "tags": "windows-10"
    }
  ]
}
```

Záver

Som spokojný, že som počas robenia tohto projektu nenarazil na väčšie problémy, avšak stále je čo zlepšiť. Tester mi zbehol vpohode, avšak nezhodujú sa časy a dátumy, a to je preto lebo som to nerobil podľa ISO8601 v UTC a nezaokrúhovaľ som časy, keďže aj v príkladoch boli raz zaokrúhlené na 3 desatinné miesta, raz na 1 desatinné miesto, a teda som si nebol istý, podľa ktorého to mám spraviť. Okrem toho si myslím, že môj program funguje a vypisuje správne.