

Database Systems

documentation for assignment 2

Teacher: William Brach

Email: william.brach@stuba.sk

Student: Alexandra Vetrov

Email: xvetrov@stuba.sk

Contents

Overview.....	3
Endpoint GET /v2/posts/:post id/users	3
Result.....	4
Endpoint GET /v2/users/:user id/friends	4
Result.....	6
Endpoint GET /v2/tags/:tagname/stats	6
Result.....	9
Endpoints GET /v2/posts/?duration=:duration in minutes&limit=:limit	9
GET /v2/posts?limit=:limit&query=:query.....	9
Results	12

Overview

Task: Basic SQL queries

Programming language: Python

In my project, I've structured the code so all the SQL queries that my HTTP endpoints need are in a file called **queries.py**. In **query.py**, I've set up the HTTP endpoint handlers using FastAPI. Inside these handlers, I call the appropriate SQL query from **queries.py** I've also created some helper functions. One of them, **to_utc_isoformat(dt: datetime) -> str**, does the heavy lifting of converting datetime objects into the ISO 8601 format with timezone information.

Endpoint GET /v2/posts/:post id/users

The generated query is executed asynchronously using `database.fetch_all()`, fetching all users related to the post from the database.

If no users are found (result is empty), it raises an `HTTPException` with a 404 status code, indicating that the post with the given ID was not found.

If users are found, each user's data is processed:

The creation date and last access date of each user are formatted into UTC ISO format using the `to_utc_isoformat()` function.

The modified user data is stored in a list called `modified_users`.

query.py

```
@router.get("/v2/posts/{post_id}/users", response_model=Root)
async def get_post_users(post_id: int):
    query = GET_POST_USERS_QUERY(post_id)
    result = await database.fetch_all(query)
    if not result:
        raise HTTPException(status_code=404, detail=f"Post with ID {post_id} not found")

    modified_users = []
    for user in result:
        user_dict = dict(user)

        if 'creationdate' in user_dict and user_dict['creationdate']:
            user_dict['creationdate'] = to_utc_isoformat(user_dict['creationdate'])

        if 'lastaccessdate' in user_dict and user_dict['lastaccessdate']:
            user_dict['lastaccessdate'] = to_utc_isoformat(user_dict['lastaccessdate'])
        modified_users.append(user_dict)

    items = [Item(**user) for user in modified_users]
    return Root(items=items)
```

Finally, the modified user data is converted into a list of `Item` objects, and these items are returned as part of the response using the `Root` model, which is specified as the response model for this endpoint.

queries.py

It selects all columns (u.*) from the users table.

It joins the users, comments, and posts tables based on certain conditions:

The comments table is joined with the users table on u.id = c.userid.

The posts table is joined with the comments table on c.postid = p.id.

It specifies a condition in the WHERE clause where the id of the posts table matches the given post_id.

The results are ordered by the creation date of the comments in descending order (ORDER BY c.creationdate DESC).

```
def GET_POST_USERS_QUERY(post_id: int) -> str:
    return f"""
SELECT u.*
FROM users u
JOIN comments c ON u.id = c.userid
JOIN posts p ON c.postid = p.id
WHERE p.id = {post_id}
ORDER BY c.creationdate DESC
"""
```

In summary, this query retrieves all user information (u.*) associated with a particular post ID (post_id). It achieves this by joining the users, comments, and posts tables and filtering the results based on the provided post_id. Finally, it orders the results based on the creation date of the comments associated with the users.

Result



```
1 {
2   "items": [
3     {
4       "id": 1865130,
5       "reputation": 1,
6       "creationdate": "2023-11-27T10:50:00.57+00:00",
7       "displayname": "flwkey12",
8       "lastaccessdate": "2023-12-03T04:23:25.53+00:00",
9       "websiteurl": "",
10      "location": null,
11      "aboutme": "",
12      "views": 5,
13      "upvotes": 0,
14      "downvotes": 0,
15      "profileimageurl": null,
16      "age": null,
17      "accountid": 25575715
18    },
19  ]
}
```

Endpoint GET /v2/users/:user id/friends

The first set selects post IDs from the posts table where the owneruserid matches the given user_id.

The second set selects post IDs from the comments table where the userid matches the given user_id, joined with the posts table on c.postid = po.id.

Then selects distinct user IDs from the comments table where the post IDs match those obtained from the postsu.

The main query selects all columns (u.*) from the users table, joining it with the commentsu on u.id = c.userid.

Results are ordered by the creation date of the users in ascending order (ORDER BY u.creationdate ASC).

queries.py

```
def GET_USERS(user_id: int) -> str:
    return f"""
    WITH postsu AS (
        SELECT po.id
        FROM posts po
        WHERE po.owneruserid = {user_id}
        UNION
        SELECT c.postid
        FROM comments c
        JOIN posts po ON c.postid = po.id
        WHERE c.userid = {user_id}
    ),
    commentsu AS (
        SELECT DISTINCT c.userid
        FROM comments c
        JOIN postsu p ON c.postid = p.id
    )
    SELECT u.*
    FROM users u
    JOIN commentsu c ON u.id = c.userid
    ORDER BY u.creationdate ASC;
    """
```

In summary, this query fetches user data for a specified user ID by first determining the posts and comments associated with the user. Then, it selects the corresponding user data and orders the results based on the users' creation dates in ascending order.

query.py

```
@router.get("/v2/users/{user_id}/friends")
async def get_status(user_id: int):
    query = GET_USERS(user_id)
    result = await database.fetch_all(query)
    if not result:
        raise HTTPException(status_code=404, detail=f"User with ID {user_id} not found")
    modified_users = []
    for user in result:
        user_dict = dict(user)
        if 'creationdate' in user_dict and user_dict['creationdate']:
            user_dict['creationdate'] =
to_utc_isoformat2(user_dict['creationdate'])
        if 'lastaccessdate' in user_dict and user_dict['lastaccessdate']:
            user_dict['lastaccessdate'] =
to_utc_isoformat2(user_dict['lastaccessdate'])
```

```

    modified_users.append(user_dict)

    json_result = {"items": modified_users}
    return json_result

```

In summary, this route retrieves user data along with their friends, formats the data, and returns it as a JSON response. It ensures that the user's creation date and last access date are in a specific format before sending the response.

Result

```

1  {
2      "items": [
3          {
4              "id": 62,
5              "reputation": 38850,
6              "creationdate": "2009-07-15T05:13:35.880+00",
7              "displayname": "Svish",
8              "lastaccessdate": "2023-10-09T16:07:45.570+00",
9              "websiteurl": "https://www.geekality.net",
10             "location": "Norway",
11             "aboutme": "<p>Software Developer, Geek, HSP, SDA, ..., ope
something non-computer-related...</p>\n\n<p>Not the best at bragging ab
12             "views": 2044,
13             "upvotes": 260,
14             "downvotes": 45,
15             "profileimageurl": null,
16             "age": null,
17             "accountid": 17529
18         },
19         {
20             "id": 666,
21             "reputation": 221,
22             "creationdate": "2009-07-15T08:48:35.320+00",
23             "displayname": "Matthew Savage",
24             "lastaccessdate": "2021-05-13T03:22:53.880+00",
25             "websiteurl": "",
26             "location": "Canberra, Australia",
27             "aboutme": "",
28             "views": 18,
29             "upvotes": 1,
30             "downvotes": 0,
31             "profileimageurl": null,
32             "age": null,
33             "accountid": 9885
34         },
35     ]
36 }

```

Endpoint GET /v2/tags/:tagname/stats

PostCounts: Counts the total number of posts for each day of the week. It selects the day of the week using the to_char() function and groups the results by the day of the week.

TaggedPostCounts: Counts the number of posts tagged with the specified tag for each day of the week. It filters posts based on the provided tag name and groups the results by the day of the week.

Main Query:

It selects the day of the week and calculates the percentage of posts tagged with the specified tag out of the total number of posts for each day of the week.

The percentage calculation is performed as follows:

$(\text{tpc.tagged_posts} * 100.0) / \text{pc.total_posts}$: Calculates the percentage of tagged posts out of total posts.

$\text{COALESCE}(\text{ROUND}(\dots), 0)$ AS percentage: Rounds the percentage to two decimal places and handles cases where the total number of posts for a specific day is zero (avoids division by zero).

The results are obtained by left joining the PostCounts and TaggedPostCounts CTEs on the day of the week.

The results are ordered by the day of the week, with Monday as the first day and Sunday as the last day.

queries.py

```
def GET_TAGS(tag: str):
    return f"""
WITH PostCounts AS (
    SELECT
        TRIM(to_char(post.creationdate, 'Day')) AS day_of_week,
        COUNT(DISTINCT post.id) AS total_posts
    FROM
        tags
    JOIN post_tags pt ON tags.id = pt.tag_id
    JOIN posts post ON pt.post_id = post.id
    GROUP BY
        day_of_week
),
TaggedPostCounts AS (
    SELECT
        TRIM(to_char(p.creationdate, 'Day')) AS day_of_week,
        COUNT(DISTINCT p.id) AS tagged_posts
    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 = '{tag}'
    GROUP BY
        day_of_week
)
SELECT
    pc.day_of_week,
    COALESCE(ROUND(((tpc.tagged_posts * 100.0) / pc.total_posts), 2), 0) AS
percentage
FROM
```

```
PostCounts pc
LEFT JOIN TaggedPostCounts tpc ON pc.day_of_week = tpc.day_of_week
ORDER BY
CASE pc.day_of_week
WHEN 'Monday' THEN 1
WHEN 'Tuesday' THEN 2
WHEN 'Wednesday' THEN 3
WHEN 'Thursday' THEN 4
WHEN 'Friday' THEN 5
WHEN 'Saturday' THEN 6
WHEN 'Sunday' THEN 7
END;
"""
```

In summary, this query retrieves statistics for each day of the week regarding the percentage of posts tagged with a specific tag out of the total number of posts. It calculates these statistics based on the posts' creation dates and their corresponding tags, presenting the data ordered by the days of the week.

query.py

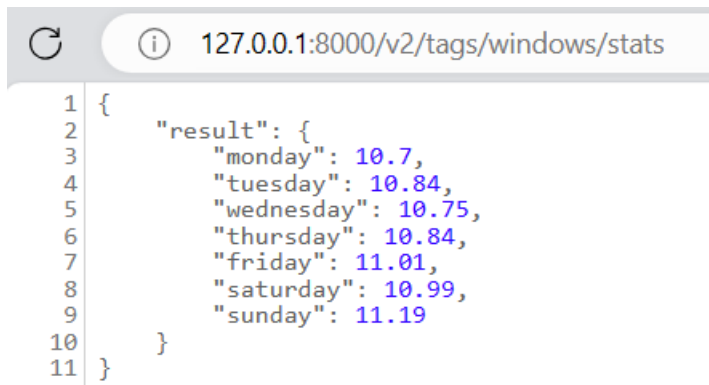
```
@router.get("/v2/tags/{tagname}/stats")
async def get_status(tagname: str):
    query = GET_TAGS(tagname)
    result = await database.fetch_all(query)
    if not result:
        raise HTTPException(status_code=404, detail=f"Tag {tagname} not found")
    stats_result = {}

    for item in result:
        day = item['day_of_week'].lower()
        percentage = item['percentage']
        stats_result[day] = percentage
    json_result = {"result": stats_result}

    return json_result
```

In summary, this route retrieves statistics for a specific tag name, processes the results, and returns them as a JSON response. If the tag name is not found, it raises an HTTPException with a 404 status code.

Result



```
1 {
2   "result": {
3     "monday": 10.7,
4     "tuesday": 10.84,
5     "wednesday": 10.75,
6     "thursday": 10.84,
7     "friday": 11.01,
8     "saturday": 10.99,
9     "sunday": 11.19
10  }
11 }
```

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

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

I also combined the 4th and 5th endpoints so that the tester could test

```
def Duration(duration_m: int, limit: int):
    return f"""
    SELECT
posts.id,posts.creationdate,posts.viewcount,posts.lasteditdate,posts.lastac
tivitydate,posts.title,
    posts.closeddate,
    ROUND(EXTRACT(EPOCH FROM posts.closeddate - posts.creationdate) / 60,
2) AS duration
FROM posts
WHERE closeddate IS NOT NULL AND EXTRACT(EPOCH FROM posts.closeddate -
posts.creationdate) / 60 < {duration_m}
ORDER BY creationdate DESC
LIMIT {limit};
    """
```

This query retrieves information about posts based on their duration and other attributes.

It selects specific attributes from the posts table.

Calculates the duration between the closeddate and creationdate of posts, rounding to two decimal places.

Filters posts where closeddate is not null and the duration is less than the provided duration_m.

Orders the results by creationdate in descending order.

Limits the number of results returned by the specified limit.

```
def search_q(limit: int, query: str):
    return f"""
    SELECT
p.id,p.creationdate,p.viewcount,p.lasteditdate,p.lastactivitydate,p.title,p
.body,p.answercount,p.closeddate,
    (SELECT array_agg(t.tagname)
```

```
FROM post_tags pt
JOIN tags t ON pt.tag_id = t.id
WHERE pt.post_id = p.id
GROUP BY pt.post_id) AS tag_array
FROM posts p
WHERE unaccent(p.body) ILIKE unaccent('%{query}%') OR unaccent(p.title)
ILIKE unaccent('%{query}%')
ORDER BY
    p.creationdate DESC
LIMIT
    {limit};
"""
```

This query retrieves posts based on a search query provided by users.

It selects specific attributes of posts and constructs an array of associated tag names.

The array_agg() function aggregates tag names into an array for each post.

It joins the posts table with the post_tags table and the tags table to get tag information.

Filters posts where the body or title matches the provided search query, using case-insensitive and accent-insensitive comparison.

Orders the results by creationdate in descending order.

Limits the number of results returned by the specified limit.

query.py

```
@router.get("/v2/posts")
async def get_posts(limit: int, query: Optional[str] = None, duration:
Optional[int] = None):
    json_result = []
    if query is not None:
        sql_query = search_q(limit, query)
    elif duration is not None:
        sql_query = Duration(duration, limit)
    else:
        raise HTTPException(status_code=400, detail="Must provide either
query or duration")

    result = await database.fetch_all(sql_query)
    if not result:
        raise HTTPException(status_code=404, detail="Not found")

    if query is not None:
        posts = {}
        for row in result:
            post_id = row['id']
            if post_id not in posts:
                posts[post_id] = {
                    "id": row['id'],
                    "creationdate": to_utc_isoformat2(row['creationdate']),
                    "viewcount": row['viewcount'],
                    "lasteditdate": to_utc_isoformat2(row['lasteditdate'])
                }
            if row['lasteditdate'] else None,
                "lastactivitydate":
                    to_utc_isoformat2(row['lastactivitydate']),
                    "title": row['title'],
                    "body": row['body'],
                    "answercount": row['answercount'],
                    "closeddate": to_utc_isoformat2(row['closeddate']) if
```

```
row['closeddate'] else None,
    "tags": row['tag_array']
}

json_result = {"items": list(posts.values())}
elif duration is not None:
    modified = []
    for post in result:
        posts_dict = dict(post)
        if 'closeddate' in posts_dict and posts_dict['closeddate']:
            posts_dict['closeddate'] =
to_utc_isoformat2(posts_dict['closeddate'])
        if 'creationdate' in posts_dict and posts_dict['creationdate']:
            posts_dict['creationdate'] =
to_utc_isoformat2(posts_dict['creationdate'])
        if 'lastactivitydate' in posts_dict and
posts_dict['lastactivitydate']:
            posts_dict['lastactivitydate'] =
to_utc_isoformat2(posts_dict['lastactivitydate'])
        if 'lasteditdate' in posts_dict and posts_dict['lasteditdate']:
            posts_dict['lasteditdate'] =
to_utc_isoformat2(posts_dict['lasteditdate'])
        modified.append(posts_dict)
    json_result = {"items": modified}

return json_result
```

This endpoint /v2/posts retrieves posts based on specified criteria such as limit, query, or duration. Here's how it works:

It accepts three parameters: limit (the maximum number of posts to retrieve), query (a string to search within post titles or bodies), and duration (the maximum duration in minutes between post creation and closure).

If the query parameter is provided, it constructs a search query using the search_q function and executes it. If the duration parameter is provided, it constructs a duration-based query using the Duration function.

If no posts are found based on the query criteria, it raises an HTTPException with a status code of 404 (Not Found).

Results

```

1 {
2   "items": [
3     {
4       "id": 1818996,
5       "creationdate": "2023-12-01T15:02:47.460+00",
6       "viewcount": 14,
7       "lasteditdate": null,
8       "lastactivitydate": "2023-12-01T15:02:47.460+00",
9       "title": "Loading html from a same-domain link via t
10      "closeddate": "2023-12-01T15:09:24.890+00",
11      "duration": 6.62
12    },
13    {
14      "id": 1818849,
15      "creationdate": "2023-11-30T15:55:32.140+00",
16      "viewcount": 22924,
17      "lasteditdate": null,
18      "lastactivitydate": "2023-11-30T15:55:32.140+00",
19      "title": "Why is my home router address is 10.x.x.x
20      "closeddate": "2023-11-30T15:59:23.560+00",
21      "duration": 3.86
22    }
23  ]
24 }

```

```

1 {
2   "items": [
3     {
4       "id": 1819160,
5       "creationdate": "2023-12-03T04:22:43.590+00",
6       "viewcount": 7,
7       "lasteditdate": null,
8       "lastactivitydate": "2023-12-03T04:22:43.590+00",
9       "title": "Keyboard not working on khali linux",
10      "body": "<p>I have recently installed virtualbox
it is not taking keyboard(Samsung bluetooth 500) input. Pleas
11      "answercount": 0,
12      "closeddate": null,
13      "tags": [
14        "virtual-machine"
15      ]
16    }
17  ]
18 }

```