

Database Systems

documentation for assignment 3

Teacher: William Brach

Email: william.brach@stuba.sk

Student: Alexandra Vetrov

Email: xvetrov@stuba.sk

Contents

Overview	3
Endpoint GET /v3/users/:user id/badge history	3
Result	5
Endpoint GET /v3/tags/:tag/comments?count=:count.....	5
Result.....	7
Endpoint GET /v3/tags/:tagname/comments/:position?limit=:limit	7
Result.....	9
Endpoints GET /v3/posts/:postid?limit=:limit.....	9
Results.....	11

Overview

Task: Advanced 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_isoformat2(dt: datetime) -> str**, does the heavy lifting of converting datetime objects into the ISO 8601 format with timezone information.

Endpoint GET /v3/users/:user id/badge history

query.py

```
@router.get("/v3/users/{user_id}/badge_history")
async def get_badges(user_id: int):
    query = first(user_id)
    result = await database.fetch_all(query)
    transformed_items = []
    position_counter = 1
    for item in result:
        created_at_p = to_utc_isoformat2(item["created_at_p"])
        created_at_b = to_utc_isoformat2(item["created_at"])
        post = {
            "id": item["id_p"],
            "title": item["title_p"],
            "type": "post",
            "created_at": created_at_p,
            "position": position_counter
        }
        transformed_items.append(post)

        badge = {
            "id": item["id_b"],
            "title": item["title_b"],
            "type": "badge",
            "created_at": created_at_b,
            "position": position_counter
        }
        transformed_items.append(badge)

        position_counter += 1

    result2 = {"items": transformed_items}
    return result2
```

Overall, this function is an API endpoint that retrieves a user's badge history from the database, transforms the data into a specific format, and returns it as a JSON response.

queries.py

```
def first(user_id: int) -> str:
    return f"""
    SELECT
    p.id AS id_p,
    p.title AS title_p,
```

```
        p.creationdate AS created_at_p,
        b.id AS id_b,
        b.name AS title_b,
        b.date AS created_at
FROM
    users u
JOIN
    posts p ON u.id = p.owneruserid
JOIN
    badges b ON u.id = b.userid
WHERE
    p.creationdate < b.date
    AND u.id = {user_id}
    AND p.creationdate = (
        SELECT MAX(p2.creationdate)
        FROM posts p2
        WHERE p2.owneruserid = u.id AND p2.creationdate < b.date
    )
    AND b.date = (
        SELECT MIN(b2.date)
        FROM badges b2
        WHERE b2.userid = u.id AND b2.date > p.creationdate
    )
    AND b.id = (
        -- Select the badge with the lowest ID in case of date tie
        SELECT b3.id
        FROM badges b3
        WHERE b3.userid = u.id AND b3.date = (
            SELECT MIN(b4.date)
            FROM badges b4
            WHERE b4.userid = u.id AND b4.date > p.creationdate
        )
        ORDER BY b3.id ASC
        LIMIT 1
    )
ORDER BY
    p.creationdate ASC, b.date ASC;
"""
```

The query selects several fields from the database tables, including post ID, post title, post creation date, badge ID, badge name, and badge date.

It joins the users table with the posts table on the condition that the user ID matches the owner user ID of the post. It joins the result of the previous join with the badges table on the condition that the user ID matches the badge user ID.

Conditions are specified to filter the rows:

The post creation date must be before the badge date. The user ID must match the input parameter `user_id`. The post creation date must be the maximum among posts created by the user before the badge date. The badge date must be the minimum among badges earned by the user after the post creation date.

In case of a tie in badge dates, the badge with the lowest ID is selected.

It specifies the order in which the results should be sorted:

Posts are sorted in ascending order by creation date.

Badges are sorted in ascending order by date.

Subquery 1:

It finds the maximum creation date of posts owned by the user (u.id) before the badge date (b.date).

Subquery 2:

It finds the minimum badge date earned by the user (u.id) after the post creation date (p.creationdate).

Subquery 3:

It selects the badge with the lowest ID among badges earned by the user (u.id) on the same date as the result of Subquery 2.

Result

```
← 127.0.0.1:8000/v3/users/120/badge_history

{
  "items": [
    {
      "id": 7744,
      "title": "How do I make Firefox remember its window size?",
      "type": "post",
      "created_at": "2009-07-18T03:33:08.597+00",
      "position": 1
    },
    {
      "id": 5453,
      "title": "Student",
      "type": "badge",
      "created_at": "2009-07-18T03:47:30.730+00",
      "position": 1
    },
    {
      "id": 8957,
      "title": null,
      "type": "post",
      "created_at": "2009-07-20T02:27:58.430+00",
      "position": 2
    },
    {
      "id": 6095,
      "title": "Teacher",
      "type": "badge",
      "created_at": "2009-07-20T02:32:30.713+00",
      "position": 2
    },
    ...
  ]
}
```

Endpoint GET /v3/tags/:tag/comments?count=:count

query.py

```
@router.get("/v3/tags/{tag}/comments")
async def get_comm(tag: str, count: Optional[int] = None):
    query, values = second(tag, count)
    result = await database.fetch_all(query, values)
    transformed_items = []
    for item in result:
        comm = {
            "post_id": item["postid"],
            "title": item["title"],
            "displayname": item["displayname"],
```

```

        "text": item["text"],
        "post_created_at":
to_utc_isoformat2(item["post_creationdate"]),
        "created_at": to_utc_isoformat2(item["comment_creationdate"]),
        "diff": format_time_diff(item["diff"]),
        "avg": format_time_diff(item["avg_diff"])
    }
    transformed_items.append(comm)
    return {"items": transformed_items}

```

This FastAPI endpoint retrieves comments associated with a specified tag. It accepts a tag name as a path parameter and an optional count parameter to specify a minimum threshold for comment counts on posts.

```

queries.py
def second(tag: str, count: int):
    query = """
SELECT
    sub.postid,
    sub.title,
    sub.displayname,
    sub.text,
    sub.post_creationdate,
    sub.comment_creationdate,
    sub.diff,
    SUM(sub.diff) OVER (PARTITION BY sub.postid ORDER BY
sub.comment_creationdate) / NULLIF(COUNT(sub.text) OVER (PARTITION BY
sub.postid ORDER BY sub.comment_creationdate), 0) AS avg_diff
FROM (
    SELECT
        post w tag.title,
        comments.text,
        post w tag.creationdate AS post_creationdate,
        comments.creationdate AS comment_creationdate,
        user c.displayname,
        comments.postid,
        comments.creationdate - COALESCE(LAG(comments.creationdate) OVER
(PARTITION BY comments.postid ORDER BY comments.creationdate),
post w tag.creationdate) AS diff
    FROM comments
    JOIN (
        SELECT p.id, p.title, u.displayname, p.creationdate
        FROM posts p
        JOIN users u ON u.id = p.owneruserid
        JOIN post_tags pt ON p.id = pt.post_id
        JOIN tags t ON t.id = pt.tag_id
        WHERE t.tagname = :tag
        AND p.id IN (
            SELECT c.postid
            FROM comments c
            GROUP BY c.postid
            HAVING COUNT(c.id) > :count
        )
    ) AS post w tag ON comments.postid = post w tag.id
    LEFT JOIN users user_c ON comments.userid = user_c.id
) AS sub
ORDER BY sub.postid, sub.comment_creationdate;
"""
    return query, {"tag": tag, "count": count}

```

It selects various fields including post ID, post title, commenter's display name, comment text, post creation date, comment creation date, difference between consecutive comment creation dates, and average difference calculated using window functions.

It pulls data from a subquery labeled as sub, which retrieves information from the comments table and other related tables.

Subquery:

The subquery selects relevant information from posts, users, and post_tags tables.

It filters posts based on the provided tag and a count threshold for comments.

Only posts with comments exceeding the specified count threshold are included.

The subquery joins with the comments table based on matching post IDs.

It performs a left join with the users table to fetch the display name of the commenter.

It calculates the difference between consecutive comment creation dates (diff) for each post using the LAG function.

It computes the average difference (avg_diff) over a window partitioned by post ID and ordered by comment creation date. This average is calculated by summing up the differences and dividing by the count of comments. The NULLIF function is used to handle the case where the count of comments is zero, avoiding division by zero errors.

The result is ordered by post ID and comment creation date.

Result

```
← ↻ ⓘ 127.0.0.1:8000/v3/tags/networking/comments?count=40 🔍 ☆
{
  "items": [
    {
      "post_id": 1034137,
      "title": "Did I just get hacked?",
      "displayname": "Jonno",
      "text": "Yeah that doesn't look too good. I'm not an expert in Linux by any means, but somethings definitely trie
      attempted to log in as root and failed. Are there any other logs in your auth.log? Any other means of remote admin? I've seen
      looks like an SSH attempt. Looks like the IPs it was downloading from are hosted in China somewhere.",
      "post_created_at": "2016-02-01T10:21:48.690+00",
      "created_at": "2016-02-01T10:25:02.610+00",
      "diff": "00:03:13.920",
      "avg": "00:03:13.920"
    },
    {
      "post_id": 1034137,
      "title": "Did I just get hacked?",
      "displayname": "David Schwartz",
      "text": "The attack actually came from China.",
      "post_created_at": "2016-02-01T10:21:48.690+00",
      "created_at": "2016-02-01T10:30:45.310+00",
      "diff": "00:05:42.700",
      "avg": "00:04:28.310"
    },
    {
      "post_id": 1034137,
      "title": "Did I just get hacked?",
      "displayname": "void",
      "text": "Yes but what is a Microsoft owned IP doing trying to breach a device across the internet?",
      "post_created_at": "2016-02-01T10:21:48.690+00",
      "created_at": "2016-02-01T10:37:58.037+00",
      "diff": "00:07:12.727",
      "avg": "00:05:23.116"
    }
  ]
}
```

Endpoint GET /v3/tags/:tagname/comments/:position?limit=:limit

query.py

@router.get("/v3/tags/{tagname}/comments/{position}")

```
async def get_comments_by_tag_position(tagname: str, position: int, limit: Optional[int] =
None):
    query, values = third(tagname, position, limit)
    result = await database.fetch_all(query, values)
    transformed_items = []
    for item in result:
        post = {
            "id": item["id"],
            "displayname": item["displayname"],
            "body": item["body"],
            "text": item["text"],
            "score": item["score"],
            "position": position
        }
        transformed_items.append(post)
    return {"items": transformed_items}
```

This FastAPI endpoint retrieves comments associated with a specific tag at a given position within their respective posts. It accepts the tag name and position as path parameters and an optional limit parameter to specify the maximum number of comments to retrieve. The function calls an SQL query function to fetch the data from the database, passing the tag name, position, and limit as parameters. It then transforms the results into a dictionary format containing information such as comment ID, commenter's display name, post body, comment text, comment score, and position within the post. Finally, it returns the transformed data as a JSON response.

queries.py

```
def third(tagname: str, position: int, limit: int):
    # return
    query = """
        SELECT c.id,u.displayname,p.body,c.text, c.score
        FROM comments c
        JOIN public.posts p ON c.postid = p.id
        JOIN public.post_tags pt ON p.id = pt.post_id
        JOIN public.tags t ON t.id = pt.tag_id
        JOIN public.users u ON u.id = c.userid
        WHERE t.tagname = :tagname
        AND (
            SELECT COUNT(*)
            FROM comments c2
            WHERE c2.postid = p.id AND c2.creationdate <= c.creationdate
            ) = :position
        GROUP BY p.id, p.creationdate, c.creationdate, c.text, c.id,
        u.displayname, c.score
        ORDER BY p.creationdate, c.creationdate
        LIMIT :limit;
    """
    return query, {"tagname": tagname, "position": position, "limit":
limit}
```

The query selects several fields including comment ID (c.id), commenter's display name (u.displayname), post body (p.body), comment text (c.text), and comment score (c.score).

The data is sourced from the comments table (c), posts table (p), post_tags table (pt), tags table (t), and users table (u).

It joins the comments table with other tables based on foreign key relationships:

comments table is joined with posts table based on postid.posts table is joined with post_tags table based on id.post_tags table is joined with tags table based on tag_id.users table is joined with comments table based on userid.

It filters the data based on the provided tagname.

It selects comments based on their position within their respective posts. This is achieved by comparing the count of comments with creation dates less than or equal to the current comment's creation date within the same post to the provided position.

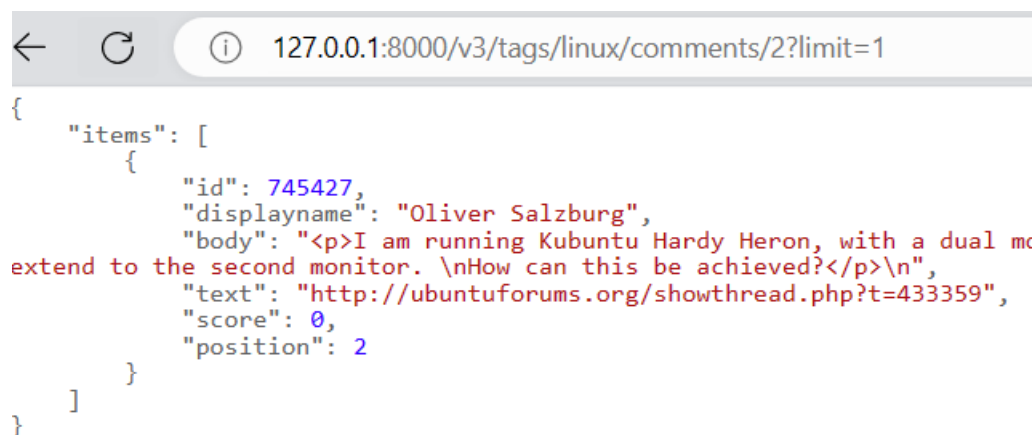
It groups the result by post ID, post creation date, comment creation date, comment text, comment ID, commenter's display name, and comment score. This ensures that each group represents a unique combination of these fields.

The result is ordered by post creation date and then by comment creation date in ascending order.

It limits the number of results returned by the query to the specified limit.

In summary, this query retrieves comments associated with a given tag, considering their position within their respective posts, and returns them in a specified order with a limit on the number of results.

Result



```
{
  "items": [
    {
      "id": 745427,
      "displayname": "Oliver Salzburg",
      "body": "<p>I am running Kubuntu Hardy Heron, with a dual m
extend to the second monitor. \nHow can this be achieved?</p>\n",
      "text": "http://ubuntuforums.org/showthread.php?t=433359",
      "score": 0,
      "position": 2
    }
  ]
}
```

Endpoints GET /v3/posts/:postid?limit=:limit

```
query.py
@router.get("/v3/posts/{postid}")
async def get_badges(postid: int, limit: int):
    query, values = fourth(postid, limit)
    result1 = await database.fetch_all(query, values)
    transformed_items = []
    for item in result1:
        created_at_p = to_utc_isoformat2(item["creationdate"])
        post = {
```

```

        "displayname": item["displayname"],
        "body": item["body"],
        "created_at": created_at_p,
    }
    transformed_items.append(post)
    if item["child_displayname"] and item["child_body"] and
item["child_creationdate"]:
        created_at_c = to_utc_isoformat2(item["child_creationdate"])
        child_post = {
            "displayname": item["child_displayname"],
            "body": item["child_body"],
            "created_at": created_at_c,
        }
        transformed_items.append(child_post)
    return {"items": transformed_items}

```

This FastAPI endpoint retrieves information about a specific post identified by its ID. It also includes child posts (comments) associated with the main post, up to a specified limit. The function calls a separate SQL query function to fetch the post data and its child posts from the database. After retrieving the data, it transforms it into a dictionary format containing details such as the display name of the post author, post body, and creation date. If available, information about child posts is also included, such as the display name of the commenter, comment body, and creation date. Finally, the function returns the transformed data as a JSON response.

```

queries.py
def fourth(postid: int, limit: int):
    query = """
SELECT u.displayname, p.body, p.creationdate, child.creationdate AS
child creationdate, child.body AS child body, child.displayname AS
child displayname
FROM posts p
LEFT JOIN users u on u.id = p.owneruserid
LEFT JOIN (
    SELECT po.body, po.parentid, po.creationdate, u.displayname
    FROM posts po
    LEFT JOIN users u on u.id = po.owneruserid
    WHERE po.parentid = :postid
    ORDER BY po.creationdate ASC
    LIMIT :limit-1
) AS child ON p.id = child.parentid
WHERE p.id = :postid
LIMIT :limit;
"""
    return query, {"postid": postid, "limit": limit}

```

The query selects various fields including the display name of the post author (u.displayname), the body of the post (p.body), the creation date of the post (p.creationdate), the creation date of child posts (child.creationdate), the body of child posts (child.body), and the display name of the commenter of child posts (child.displayname).

The data is sourced from the posts table (p) and the users table (u). It performs left joins with the users table based on the owneruserid to retrieve the display name of the post author and the commenter of child posts.

Subquery for Child Posts:

It includes a subquery labeled as child to fetch information about child posts. This subquery selects the body, parent ID, creation date, and display name of child posts. It filters child posts

based on the provided postid (ID of the main post) and orders them by creation date in ascending order.

It limits the number of child posts retrieved by the specified limit-1.

Main Query:

The main query selects data from the posts table.

It performs a left join with the child subquery based on the parentid to retrieve information about child posts associated with the main post. It filters the main post based on the provided postid. It limits the total number of results returned by the query to the specified limit.

Results

```
← ↻ ⓘ 127.0.0.1:8000/v3/posts/2154?limit=2
{
  "items": [
    {
      "displayname": "Eugene M",
      "body": "<p>So, I'm a technology guy and sometimes I have to troubleshoot a ho
actual internet connection. Usually after that point I just reset the router( and possibly
of issue I could possibly be fixing by resetting the router.</p>\n\n<p>EDIT: Just to clari
reset(paperclip in the hole) is useful. So the more accurate term would probably be restar
bother messing around with stuff if I can't make a connection to the internet at all.</p>\n
      "created_at": "2009-07-15T12:51:57.340+00"
    },
    {
      "displayname": "Ólafur Waage",
      "body": "<p>Every router has it's original firmware stored somewhere on it.</p>
usually is fixing the problem is that the config is overwritten with the original one. But
      "created_at": "2009-07-15T12:54:48.507+00"
    }
  ]
}
```