DBMS Final Project Report

Team Members

Satyadev Subudhi : 112101058
Vijay : 112101060
Maaya.R.Srivatsav:112101055

INDEX

Relational design of the database

Schemas :

Users :

Attributes : (UserID , Username, Score, views ,down_votes,up_votes, User_Bio, Location, creation_date,last_access_date)

Primary Key : UserID

Field Name	Datatype	Constraints
UserID	integer	**Primary Key**
Username	text	NOT NULL
Score	integer	Default =0
Views	integer	Default=0
down_votes	integer	Default=0
up_votes	integer	Default=0
User_Bio	text	
Location	text	
creation_date	TIMESTAMP	NOT NULL
last_access_date	TIMESTAMP	NOT NULL

Attributes :

- UserID: This is the primary key of the table. Unique Id of a user, assigned serially in order of insertion.
- Username : name of the user
- score: It is a measure of the users activity, can also be viewed as trust points. It is earned and lost based on user activity. A user gains 10 points when his question is voted up or his answer is voted up. Also a user loses 2 points when his question or answer is voted down and loses 1 point if the user downvotes a question or answer.

- views: Total number of times the posts of that user has been viewed.
- down_votes: Total number of downvotes the user has given (to a question, answer or comment).
- up_votes: Total number of upvotes the user has given (to a question, answer or comment).
- location: Most recent location of the user.
- User_Bio: Optional Details about the user that he/she wants to reveal.
- creation_date: Non-Null timestamp representing account creation date and time.
- last_access_date: Non-Null timestamp representing date and time when user last accessed the website.

```
                                                Table "public.users"
      Column      |            Type             | Collation | Nullable |                Default
------------------+-----------------------------+-----------+----------+----------------------------------------
 user_id          | integer                     |           | not null | nextval('users_user_id_seq'::regclass)
 user_name        | character varying(255)      |           | not null |
 score            | integer                     |           | not null | 0
 views            | integer                     |           |          | 0
 down_votes       | integer                     |           |          | 0
 up_votes         | integer                     |           |          | 0
 location         | character varying(512)      |           |          |
 user_bio         | text                        |           |          |
 creation_date    | timestamp without time zone |           | not null |
 last_access_date | timestamp without time zone |           | not null |
```

```
Indexes:
    "users_pkey" PRIMARY KEY, btree (user_id)
    "user_id_index" hash (user_id)
    "user_index4" btree (score)
Check constraints:
    "users_check" CHECK (creation_date <= last_access_date)
Referenced by:
    TABLE "badges" CONSTRAINT "badges_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(user_id) ON UPDATE CASCADE ON DELETE C
DE
    TABLE "comments" CONSTRAINT "comments_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(user_id) ON UPDATE CASCADE
    TABLE "posts" CONSTRAINT "posts_owner_user_id_fkey" FOREIGN KEY (owner_user_id) REFERENCES users(user_id) ON UPDATE CASCADE
    TABLE "votes" CONSTRAINT "votes_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(user_id) ON UPDATE CASCADE ON DELETE SET
L
Triggers:
    del_user BEFORE DELETE ON users FOR EACH ROW EXECUTE FUNCTION check_upd_user()
    insert_badge AFTER INSERT ON users FOR EACH ROW EXECUTE FUNCTION insert_badge_user()
```

Question (or) Posts :

Attributes : (post_id, owner_user_id, post_type, best_answer_id, score, parent_id, view_count Answer_count, comment_count, owner_username, tags, body,Upvotes, Downvotes,creation date)

Primary key : id
Foreign Key : owner_user_id references Users , post_id references Votes

Field Name	Datatype	Constraints
post_id	integer	**Primary key**
owner_user_id	integer	**Foreign key** referencing Users table
post_type_id	integer	NOT NULL (check constraint value only in (1,2)
best_answer_id	integer	**Foreign ke**y referencing Posts table itself (gives id of best answer from Posts table based on score)
score	integer	Default=0
parent_id	integer	**Foreign key** referencing Posts table itself (in case post type is answer (2) then it gives id of question post)
answer_count	integer	Default=0
comment_count	integer	Default=0
tags	text	
body	text	
Upvotes	integer	Default=0
Downvotes	integer	Default=0
creation_date	TIMESTAMP	NOT NULL

Attributes :

- post_Id: This is the primary key of the table. Assigned in serial order in order of creation.
- owner_user_id: The id of the owner who wrote the post.
- post_type_id: Stores integers depending on the kind of post it is (1:question post, 2:answer post).
- best_answer_id: Only present if post_type_id=1 (A question post), this will store the id of the post which the original poster recognizes as the best answer(that it the one with most upvotes).
- score: A score is essentially the difference between the number of upvotes and the number of downvotes that a particular post has.
- parent_id: Only present if the post_type_id=2 (An answer post), in which case it stores the post id of the parent post i.e question post.
- answer_count: Number of answers (only populated for question posts)
- comment_count: Number of comments on a post
- tags: Tag names of all the marked tags of that particular post.
- Upvotes: no of upvotes for the question/answer
- downvotes: no of downvotes for the question/answer
- body: The content of the post is stored as text
- creation_date: Non-Null timestamp representing account creation date and time.

Table "public.posts"

Column	Type	Collation	Nullable	Default
post_id	integer		not null	nextval('posts_post_id_seq'::regclass)
owner_user_id	integer			
post_type_id	smallint		not null	
best_answer_id	integer			
score	integer		not null	0
parent_id	integer			
answer_count	integer			0
comment_count	integer			0
tags	character varying(512)			
body	text			
creation_date	timestamp without time zone		not null	
upvotes	integer			0
downvotes	integer			0

```
Indexes:
    "posts_pkey" PRIMARY KEY, btree (post_id)
    "user_id_index2" hash (post_id)
Check constraints:
    "check_type_values" CHECK (post_type_id = ANY (ARRAY[1, 2]))
Foreign-key constraints:
    "posts_best_answer_id_fkey" FOREIGN KEY (best_answer_id) REFERENCES posts(post_id) ON UPDATE CASCADE ON DELETE SET NULL
    "posts_owner_user_id_fkey" FOREIGN KEY (owner_user_id) REFERENCES users(user_id) ON UPDATE CASCADE
    "posts_parent_id_fkey" FOREIGN KEY (parent_id) REFERENCES posts(post_id) ON UPDATE CASCADE ON DELETE CASCADE
Referenced by:
    TABLE "comments" CONSTRAINT "comments_post_id_fkey" FOREIGN KEY (post_id) REFERENCES posts(post_id) ON UPDATE CASCADE ON DE
ASCADE
    TABLE "posts" CONSTRAINT "posts_best_answer_id_fkey" FOREIGN KEY (best_answer_id) REFERENCES posts(post_id) ON UPDATE CASCA
DELETE SET NULL
    TABLE "posts" CONSTRAINT "posts_parent_id_fkey" FOREIGN KEY (parent_id) REFERENCES posts(post_id) ON UPDATE CASCADE ON DELE
CADE
    TABLE "votes" CONSTRAINT "votes_post_id_fkey" FOREIGN KEY (post_id) REFERENCES posts(post_id) ON UPDATE CASCADE ON DELETE C
Triggers:
    del_post BEFORE DELETE ON posts FOR EACH ROW EXECUTE FUNCTION check_del_post()
    upd_ans_ct AFTER INSERT ON posts FOR EACH ROW EXECUTE FUNCTION check_ans_ct()
    upd_ans_ct_del AFTER DELETE ON posts FOR EACH ROW EXECUTE FUNCTION check_del_ans_ct()
    upd_tag_ct AFTER INSERT ON posts FOR EACH ROW EXECUTE FUNCTION check_tag_ct()
    vote_update AFTER INSERT ON posts FOR EACH ROW EXECUTE FUNCTION vote_update_trig()
```

Votes :

Attributes : (vote_id ,vote_type_id, user_ID, post_ID)

Primary Key : vote_id

Foreign Key : user_ID references Users , post_ID references Posts

Field Name	Datatype	Constraints
vote_id	integer	**Primary Key**
vote_type_id	integer	NOT NULL (check constraint vote_type_id in (2,3))
user_id	integer	**Foreign ke**y referencing Users table
post_id	integer	NOT NULL , **Foreign key** references Posts table (on delete cascade on update cascade)
creation_date	TIMESTAMP	NOT NULL

Attributes :

- Vote_id : This is the primary key of votes table that uniquely identifies its records. Its domain is integer
- Vote_type : This field tells whether a given vote is an upvote(1) or a downvote(2)
- User_id : Id of the user who has upvoted / downvoted that post. (foriegn key referencing Users)
- Post_id : id of the post to which the vote was casted. If the post is deleted then this field should also be deleted , hence the usage of cascade
- creation_date: Non-Null timestamp representing account creation date and time.

Table "public.votes"

Column	Type	Collation	Nullable	Default
vote_id	integer		not null	nextval('votes_vote_id_seq'::regclass)
user_id	integer			
post_id	integer		not null	
vote_type_id	smallint		not null	
creation_date	timestamp without time zone		not null	

```
Indexes:
    "votes_pkey" PRIMARY KEY, btree (vote_id)
    "user_id_index3" hash (vote_id)
Check constraints:
    "vote_type_id" CHECK (vote_type_id = ANY (ARRAY[3, 2]))
Foreign-key constraints:
    "votes_post_id_fkey" FOREIGN KEY (post_id) REFERENCES posts(post_id) ON UPDATE CASCADE ON DELETE CASCADE
    "votes_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(user_id) ON UPDATE CASCADE ON DELETE SET NULL
Triggers:
    best_ans_upd AFTER INSERT ON votes FOR EACH ROW EXECUTE FUNCTION check_best_ans()
    vote_update AFTER INSERT ON votes FOR EACH ROW EXECUTE FUNCTION vote_update_trig()
```

Comments :

Attributes : (comment_id, user_id, post_id, score, username, comment_text)

Primary Key : comment_id

Foreign Key : user_id references Users , post_id references posts , comment_id references Votes

Field Name	Datatype	Constraints
comment_id	integer	**Primary Key**
user_id	integer	NOT NULL, **Foreign key** referencing Users table
post_id	smallint	NOT NULL,**Foreign key** referencing Posts table
comment_text	text	

Attributes :

- comment_id : This is the primary key of Comments table that uniquely identifies its records. Its domain is integer
- user_id : id of the user who casted the comment (foreign key)
- post_id : Id of the post to which the comment has been addressed to
- Comment_text : The body of the comment.

```
project=# \d comments
                                    Table "public.comments"
    Column     |  Type   | Collation | Nullable |                   Default
---------------+---------+-----------+----------+---------------------------------------------
 comment_id    | integer |           | not null | nextval('comments_comment_id_seq'::regclass)
 post_id       | integer |           | not null |
 user_id       | integer |           |          |
 comment_text  | text    |           |          |
```

```
Indexes:
    "comments_pkey" PRIMARY KEY, btree (comment_id)
Foreign-key constraints:
    "comments_post_id_fkey" FOREIGN KEY (post_id) REFERENCES posts(post_id) ON UPDATE CASCADE ON DELETE CASCADE
    "comments_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(user_id) ON UPDATE CASCADE
Triggers:
    upd_comm_ct AFTER INSERT ON comments FOR EACH ROW EXECUTE FUNCTION check_comm_ct()
    upd_del_comm_ct AFTER DELETE ON comments FOR EACH ROW EXECUTE FUNCTION check_del_comm_ct()
```

Tags :

Attributes : (tag_id, tag_desc,count).

Primary Key : tag_id

Field Name	Datatype	Constraints
tag_id	integer	**Primary Key**
tag_desc	text	NOT NULL
count	integer	Default = 0

Attributes :

- tag_id : This is the primary key of Tags table that uniquely identifies its records. Integer datatype is used to capture the id of a tag.
- tag_desc : Text explaining what the tag is about.
- count : Number of posts containing the tag. Default value = 0.

```
                                        Table "public.tags"
  Column  |          Type          | Collation | Nullable |                Default
----------+------------------------+-----------+----------+-------------------------------------
 tag_id   | integer                |           | not null | nextval('tags_tag_id_seq'::regclass)
 tag_desc | character varying(255) |           | not null |
 count    | integer                |           |          | 0
Indexes:
    "tags_pkey" PRIMARY KEY, btree (tag_id)
```

Badges :

Attributes : (badge_id, user_id, class).
Primary Key : badge_id
Foreign Key : user_id references Users

Field Name	Datatype	Constraints
badge_id	integer	**Primary Key**
user_id	integer	NOT NULL, **Foreign key** referencing Users table (on delete cascade , on update cascade)
class	integer	NOT NULL check constraint class in (1,2,3)

Attributes :

- badge_id : This is the primary key of Badges table . Integer datatype is used to capture the id of a badge.
- user_id : id of the user to whom the badge has been assigned to.
- class : Text describing about the badge.

```
                                    Table "public.badges"
 Column  |   Type   | Collation | Nullable |                 Default
---------+----------+-----------+----------+-----------------------------------------
 badge_id | integer  |           | not null | nextval('badges_badge_id_seq'::regclass)
 user_id  | integer  |           | not null |
 class    | smallint |           | not null |
Indexes:
    "badges_pkey" PRIMARY KEY, btree (badge_id)
Check constraints:
    "badges_class_check" CHECK (class = ANY (ARRAY[1, 2, 3]))
Foreign-key constraints:
    "badges_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(user_id) ON UPDATE CASCADE ON DELETE CASCADE
```

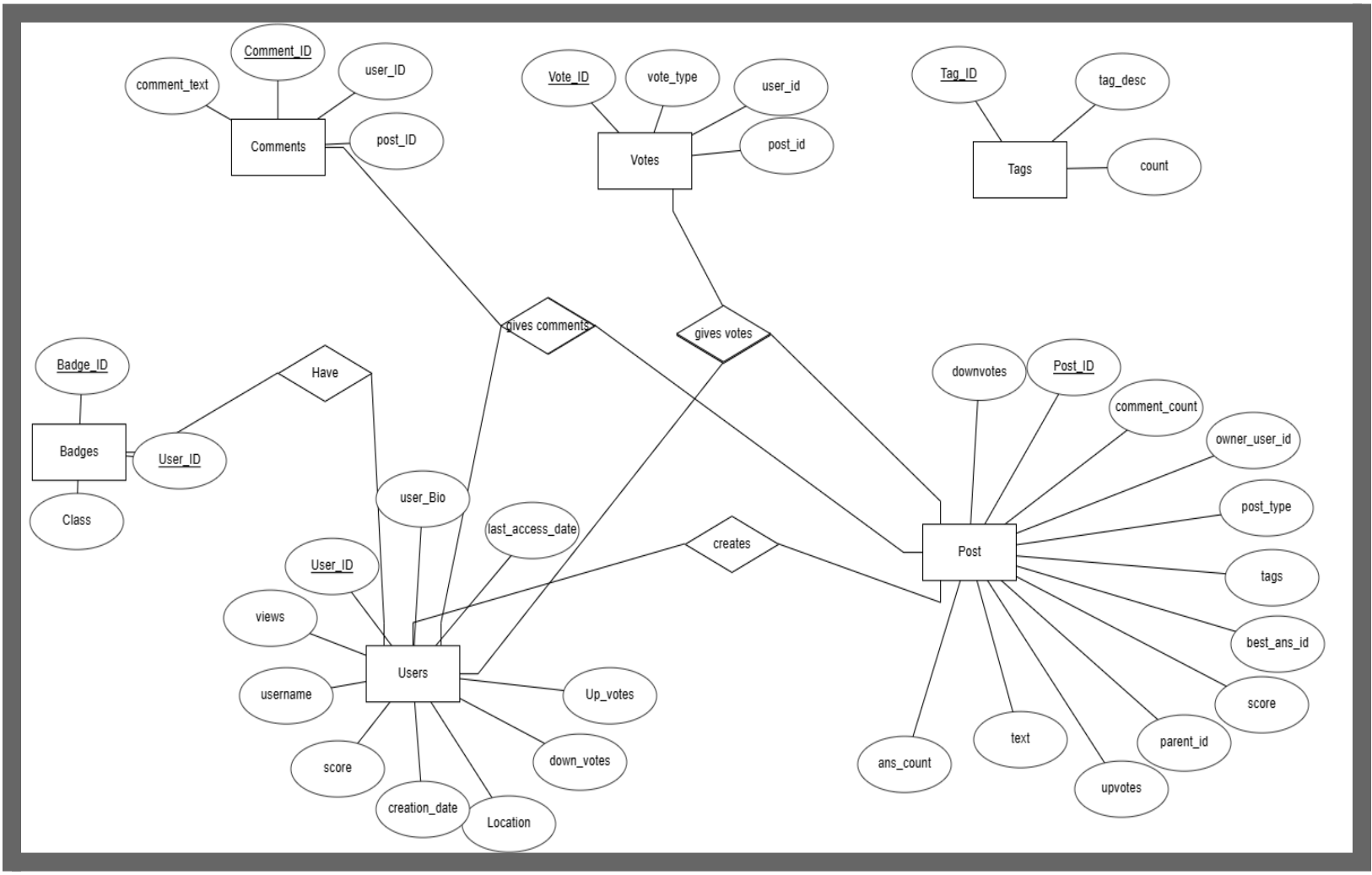
Features/Requirements of the Project

- Users will be able to view details of other users, all the posts, all the tag descriptions and leaderboard rankings.
- Users can form a new account (by inserting into users table), and can also delete or update their account- they can only delete an account created by them.
- Users can insert new posts - which can be questions or answers . They can also delete posts provided that it was posted by them.They can also up_vote or down_vote posts.While inserting the information for posts, if that tag is not in the tag_desc of the tags table, then it is added.
- Users can insert their comments for a post.

- Managers have select access to all table.
- Moderator has delete access to users and posts table. They can delete a post or ban a user if they have a score below a certain threshold.
- Leader board manager has update access to badges.This manager does a regular update of the leaderboard, based on the scores of the users.

- The best_answer_ID,answer_count,comment_count are all updated using triggers.
- When a person votes for a post, there will be a change in the score of certain users , score of the post,up_votes and down_votes count of that user,which are updated using triggers.
- Based on the above requirements, we - Have formed the most relevant tables and functions that mimics a social discussion platform and are sufficient to provide the core and fundamental functionalities of the same - Added the proper integrity constraints to the tables, also used several checks in the functions, procedure which ensures the proper functioning of our database. - Introduced different roles which have different privileges over our database.

ER DIAGRAM


Comment_ID
comment_text
user_ID
Comments
post_ID
Vote_ID
vote_type
user_id
Votes
post_id
Tag_ID
tag_desc
Tags
count
gives comments
gives votes
Badge_ID
Have
Badges
User_ID
Class
user_Bio
last_access_date
creates
User_ID
views
Users
username
Up_votes
score
down_votes
creation_date
Location
downvotes
Post_ID
comment_count
owner_user_id
post_type
Post
tags
best_ans_id
score
parent_id
text
ans_count
upvotes

Functions and Procedures

1) PROCEDURE : create_new_user , create_user_role

This procedure takes as arguments the password that the user wants to set for his account, by default the username will be user_<user_id> assigned to user.It will also take in the username,location details,user_bio details which are optional information.

This procedure will in turn call the create_user_role procedure which will create a user role with the above mentioned password.

The details of the user will also be mentioned in the users table.

Query:

```
--PROCEDURE TO CREATE NEW_USER INSERTION INTO USER TABLE.
create or replace procedure create_new_user (pwd text,us_name text,
                                             loc text DEFAULT 'UTC',
                                             us_bio text DEFAULT NULL)
language plpgsql
as $$
DECLARE
    role_name TEXT;us_id int;
BEGIN
 us_id := (select max(user_id) from users)+1;
 call create_user_role(us_id,pwd);
 insert into users(user_id,user_name,location,user_bio,creation_date,last_access_date)
 values (us_id,us_name,loc,us_bio,current_date,current_date);
end;
$$;
```

```
create or replace procedure create_user_role ( user_id int, pswd text)
language plpgsql
as $$
DECLARE
    role_name TEXT;
BEGIN
    role_name := 'user_' || user_id::text;
 IF NOT EXISTS (SELECT 1 FROM pg_roles WHERE rolname = role_name) THEN
        EXECUTE 'CREATE ROLE ' || quote_ident(role_name) || ' LOGIN PASSWORD ' || quote_literal(pswd);
        EXECUTE 'grant client_user to ' || quote_ident(role_name) ;
        RAISE NOTICE 'Role % created successfully.', role_name;
 ELSE
        RAISE NOTICE 'Role % already exists.', role_name;
 END IF;
end;
$$;
```

OUTPUT:

```
call create_new_user ('test_password','Black Adam','US','testing_data');
```

Data Output | Messages | Notifications

```
NOTICE:  Role user_121941 created successfully.
CALL

Query returned successfully in 55 msec.
```

project/user_121941@PostgreSQL 16

- project/postgres@PostgreSQL 16
- project/user_100174@PostgreSQL 16
- project/user_121940@PostgreSQL 16
- ✓ project/user_121941@PostgreSQL 16
- < New Connection... >

```
635  select * from users where user_id=121940;
```

Data Output Messages Notifications

	user_id [PK] integer	user_name character varying (255)	score integer	views integer	down_votes integer	up_votes integer	location character varying (512)
1	121940	Black Adam	0	0	0	0	US

2) PROCEDURE : create_vote(p_id int, vot_type int)

This procedure takes as arguments the post_id for which a user wants to vote for and vote_type that is 2 for up_vote and 3 for down_vote.

The details of the votes will be inserted in the votes table.

```
--PROCEDURE TO INSERT VOTES
create or replace procedure create_vote(p_id int, vot_type int)
language plpgsql
as $$
DECLARE
   us_id int;v_id int;
BEGIN
 us_id := (SELECT SUBSTRING(current_user FROM POSITION('_' IN current_user) + 1))::int;
  v_id := (select max(vote_id) from votes)+1;

 insert into votes values (v_id,us_id,p_id,vot_type,CURRENT_TIMESTAMP AT TIME ZONE 'UTC');
end;
$$;
```

OUTPUT:

```
267 select upvotes,downvotes from posts where post_id=197;
```

Data Output Messages Notifications

	upvotes integer	downvotes integer
1	6	0

project/user_100007@PostgreSQL 16

```
call create_vote(197,3)
select * from votes where post_id= 197
```

Output Messages Notifications

vote_id [PK] integer	user_id integer	post_id integer	vote_type_id smallint	creation_date timestamp without time zone
200	1595	197	2	2010-08-18 00:00:00
2332	14112	197	2	2012-10-06 00:00:00
8724	12774	197	2	2017-04-20 00:00:00
12812	100140	197	2	2024-04-28 15:55:41.764441
12813	100140	197	2	2024-04-28 15:57:27.113539
12814	100007	197	2	2024-04-30 10:08:53.112816
12815	100007	197	3	2024-04-30 10:13:29.572329

```
267  select upvotes,downvotes from posts where post_id=197;
```

Data Output Messages Notifications

	upvotes integer	downvotes integer
1	6	1

3)PROCEDURE: **create_new_post**

This procedure takes as arguments the post_type (1-question, 2- answer) , text for the post, parent_id(if its an answer post, the post_id for which it is the answer), tag associated with the post.The details of the post will be inserted in the posts table.

QUERY

project/user_121941@PostgreSQL 16

```
--PROCEDURE TO CREATE NEW_post: INSERTION INTO POSTS TABLE.

create or replace procedure create_new_post (p_type int,pos_body text DEFAULT NULL,
                                             par_id int default NULL, tag text DEFAULT NULL)
language plpgsql
as $$
DECLARE
    role_name TEXT;p_id int;us_id int;us_name text;ans_ct int;
BEGIN
 p_id := (select max(post_id) from posts)+1;
 us_id := (SELECT SUBSTRING(current_user FROM POSITION('_' IN current_user) + 1))::int;
 raise notice '% ', us_id;
 if (p_type = 1) then
 insert into posts(post_id,owner_user_id,post_type_id,parent_id,answer_count,
                   comment_count,tags, body, creation_date)
 values (p_id,us_id,p_type,par_id,0,0,tag,pos_body,current_date);
 elsif(p_type = 2) then
 insert into posts(post_id,owner_user_id,post_type_id,parent_id,answer_count,
                   comment_count,tags, body, creation_date)
 values (p_id,us_id,p_type,par_id,NULL,0,tag,pos_body,current_date);
 end if;
end;
$$;
```

OUTPUT:

```
call create_new_post (2,'Black Adam answer',3095,'newtag');
select * from posts where post_id=3096;
```

Output Messages Notifications

post_id [PK] integer	owner_user_id integer	post_type_id smallint	best_answer_id integer	score integer	parent_id integer	answer_count integer	comment_count integer	tags character
3096	121941	2	[null]	0	3095	[null]	0	newtag

4) PROCEDURE : create_new_comment(p_id int, body text DEFAULT NULL)

This procedure takes as arguments the post_id for which the comment is for, comment text of the comment. The details of the comment will be inserted in the comments table.

QUERY:

project/user_121941@PostgreSQL 16

```
create or replace procedure create_new_comment(p_id int, body text DEFAULT NULL)
language plpgsql
as $$
DECLARE
   us_id int;c_id int;
BEGIN
 c_id := (select max(comment_id) from comments)+1;
 us_id := (SELECT SUBSTRING(current_user FROM POSITION('_' IN current_user) + 1))::int;
 insert into comments values (c_id,p_id,us_id,body);
end;
$$;
```

OUTPUT:

```
call create_new_comment (11,'Black Adam');
select * from comments where comment_text = 'Black Adam'
```

Data Output Messages Notifications

	comment_id [PK] integer	post_id integer	user_id integer	comment_text text
1	8181	11	121941	Black Adam

5)

PROCEDURE : delete_post(p_id int)

This procedure takes as arguments the post_id for which is to be deleted.This record will then be deleted from the posts table. This will also activate triggers which will reduce the answer count provided the one to be deleted is an answer post.This is protected by del_post trigger that ensures that only the person who posted the post can delete it.

project/user_121941@PostgreSQL 16

QUERY:

```
create or replace procedure delete_post(p_id int)
language plpgsql
as $$
DECLARE
   us_id int;c_id int;
BEGIN
    delete from posts where post_id=p_id;
end;
$$;
```

OUTPUT:

```
call delete_post(3096);
select * from posts where post_id=3096;

select * from comments
```

Output Messages Notifications

post_id [PK] integer	owner_user_id integer	post_type_id smallint	best_answer_id integer	score integer	parent_id integer	answe intege

6) **PROCEDURE: ban_users() :**

This procedure is done by the moderator manager . This manager can ban all the user who has a score of threshold less then -50.Basically the user_bio is set to 'user_id_to_be_deleted' and then the superuser will take care of the deletion of roles.

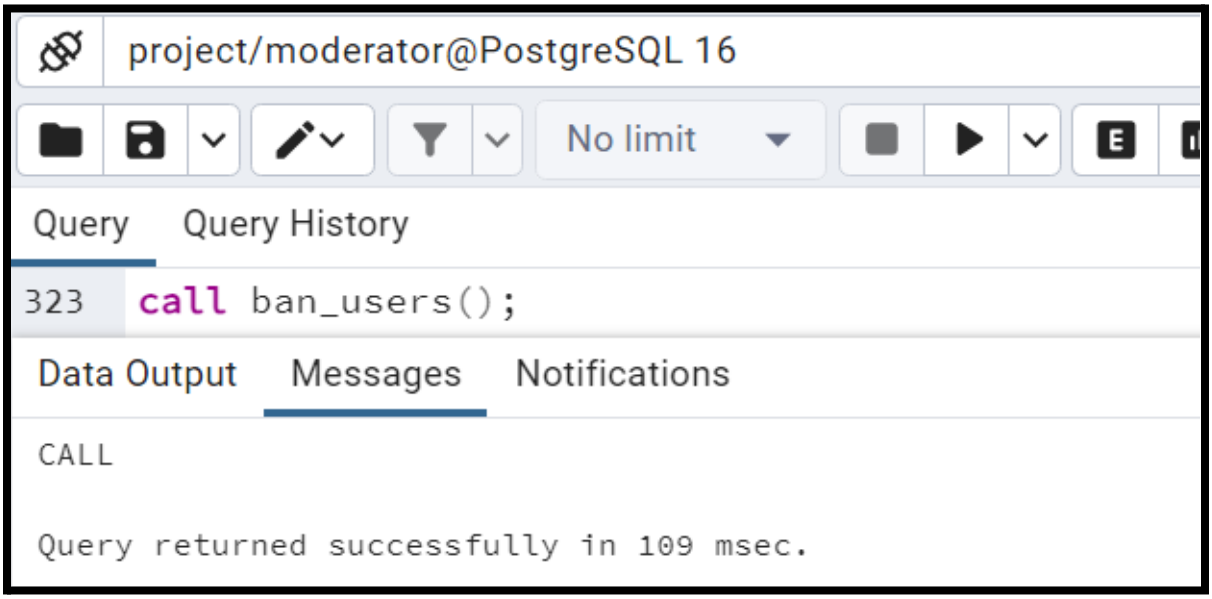
QUERY:

```
create or replace procedure ban_users()
language plpgsql
as $$
DECLARE
    role_name TEXT; i int; tt text;
Begin
if (current_user != 'moderator') then
 raise notice ' dont have authorisation for the procedure';
 else
 FOR i IN (select user_id from users) LOOP

        if ((select score from users where user_id=u_id) < -50) then

            update users set user_bio='user_id_to_be_deleted' where user_id=i;

    END IF;
    END LOOP;
    end if;
end;
$$;
```

OUTPUT:

```
project/user_100007@PostgreSQL 16
```

```
NOTICE:  not authorised procedure
CALL

Query returned successfully in 29 msec.
```



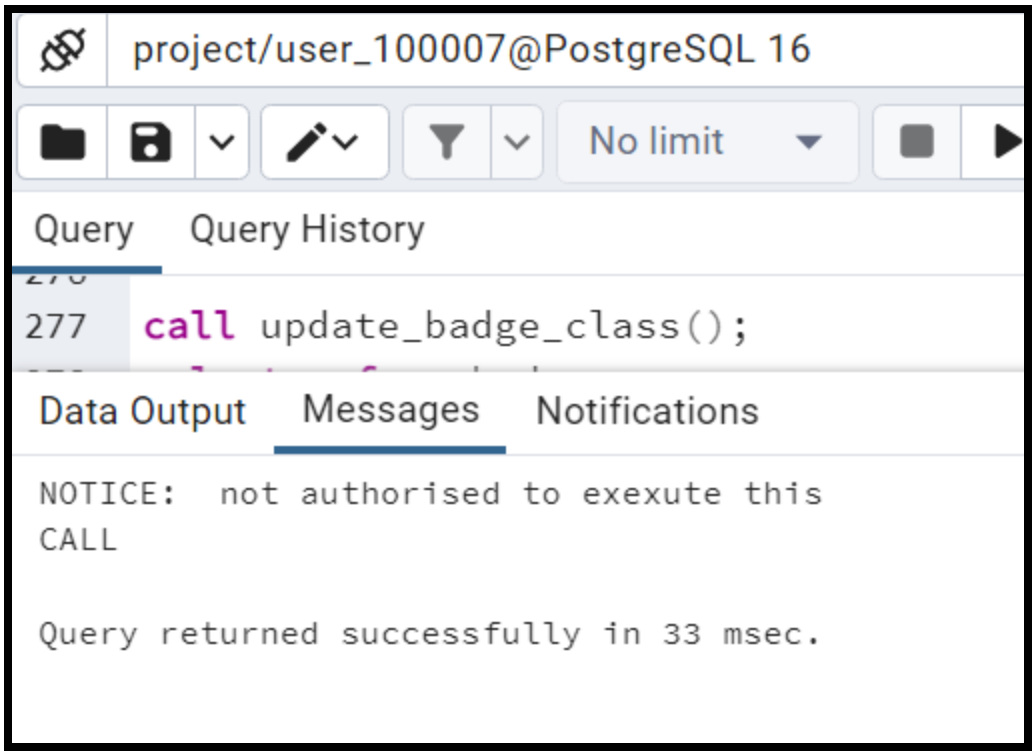
7) FUNCTION : **update_badge_class( ) :**

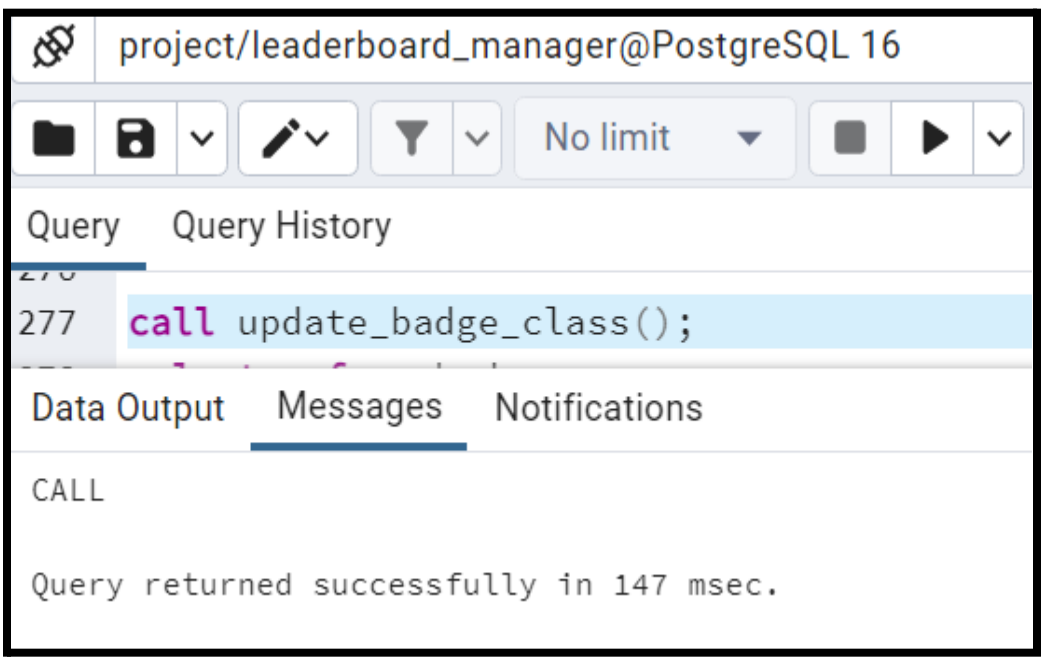
This function can only be executed by the leaderboard_manager. This is a regular updation of the leaderboard, to classify users as class 1 or 2 or 3.

QUERY

```
--UPDATE BADGES TABLE: FIRST 20 SCORES = CLASS 1, ABOVE AVG: CLASS 2, BELOW AVG: CLASS 3
create or replace procedure update_badge_class()
language plpgsql
as $$
begin
if(current_user !='leaderboard_manager') then
    raise notice 'not authorised to exexute this';
else
    Update badges set class = 1 where user_id in
    (select user_id from users where score in (select score from users order by score desc limit 20) );
    Update badges set class = 2 where user_id in
    (select user_id from users where score in (select score from users order by score desc offset 20) );

    Update badges set class = 3 where user_id in
    (select user_id from users where score<(select avg(score) from users));
end if;
end;
$$;
```

OUTPUT:





8) PROCEDURE : del_users (i int) and to_be_deleted() :

This procedure takes as input the user_id of the user whose account is to be deleted, it checks if the user_id of current user and the argument matches, if

yes, it sets all the values for that user to NULL.Currently we have given only postgres the power to drop a role. So if it finds the user_bio as user_id_to_be_deleted , it will drop the role.

```
create or replace procedure del_users(i int)
language plpgsql
as $$
DECLARE
    role_name TEXT; tt text;
Begin

       role_name := 'user_' || i::text;
        IF(current_user!=role_name) then
            raise notice 'You cant delete other accounts';
        else
            update users set user_bio='user_id_to_be_deleted' where user_id=i;

        end if;

end;
$$;
```

```
create or replace procedure to_be_deleted()
language plpgsql
as $$
DECLARE
    role_name TEXT; tt text;i int;
Begin
        if(current_user != 'postgres') then
            raise notice 'not authenticated procedure for you';
        else
        FOR i in (select user_id from users) loop
          if ((select user_bio from users where user_id=i) = 'user_id_to_be_deleted' ) then
                tt := 'user_' || i::text;
                EXECUTE 'drop role ' || quote_ident(tt);
        end if;
        end loop;
        end if;

end;
$$;
call to_be_deleted()
```

OUTPUT:

project/user_121941@PostgreSQL 16

No limit

Query Query History

```
$$;
call del_users(121940);
```

Data Output Messages Notifications

```
NOTICE:  You cant delete other accounts
CALL

Query returned successfully in 38 msec.
```

```
select user_bio from users where user_id=121941;
```

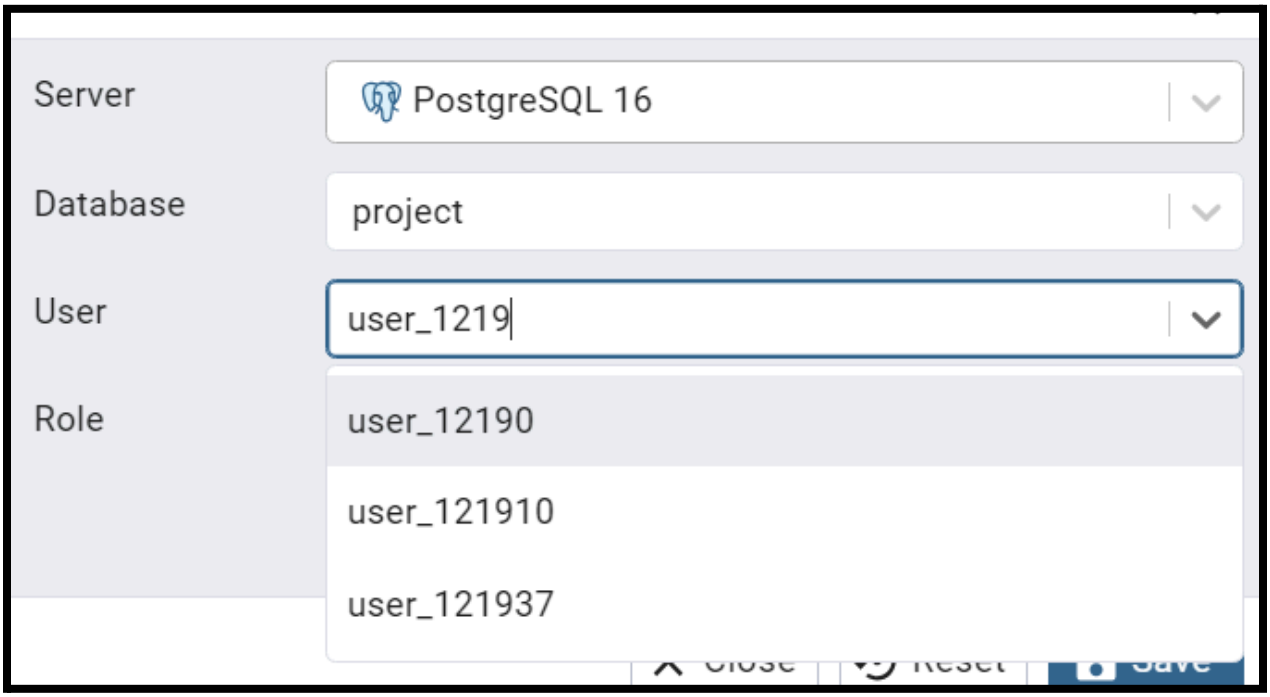
Output Messages Notifications

user_bio text
user_id_to_be_deleted

```
call to_be_deleted()
```

Data Output Messages Notifications

```
CALL

Query returned successfully in 63 msec.
```



UTILITY FUNCTIONS

9) FUNCTION: get_ans_of_post(p_id int)

This function takes as argument the post id of the post and if it is a question post , returns all the answer post available for this post, else if it is an answer post , it raises a notice.

QUERY

```
--FUNCTION TO GET_ALL ANSWERS TO A QUESTION POST
create or replace function get_ans_of_post(p_id int)
returns SETOF posts
language plpgsql

as $$
DECLARE
   us_id int;c_id int;
BEGIN
 if ((select post_type_id from posts where post_id = p_id)!=1) then
 raise exception 'Given post is an answer post';
 end if;
 return query
 select * from posts where parent_id=p_id;
end;
$$;
```

OUTPUT

```
select get_ans_of_post (11);
```

Output Messages Notifications

get_ans_of_post posts
(58,366,2,,3,11,,4,," Ganesh Sittampalam
(12,91,2,,6,11,,0,," Chris W. Rea
(57,815,2,,3,11,,0,," Alex B
(67,312,2,,3,11,,0,," C. Ross
(68,868,2,,3,11,,2,,"<h2>George <a href=""https://money.meta.stackexchar
(29,312,2,,5,11,,1,," MrChrister
(13,91,2,,3,11,,0,," Zephyr

```
396  select get_ans_of_post (493);
```

Data Output Messages Notifications

```
ERROR:  Given post is an answer post
CONTEXT:  PL/pgSQL function get_ans_of_post(integer) line 6 at RAISE

SQL state: P0001
```

10) FUNCTION get_user_info

This function takes as argument the user id of the user and returns the user info.

```
create or replace function get_user_info(u_id int)
returns SETOF users
language plpgsql

as $$

BEGIN

 return query
 select * from users where user_id=u_id;
end;
$$;
```

OUTPUT:

```
414  select get_user_info (213);
415
```

Data Output Messages Notifications

	get_user_info users
1	(213,"Jim G.",0,0,1,2,,,"2010-01-24 01:02:44","2014-11-22 09:04:13.04...

11) FUNCTION get_tag_posts

This function takes as argument the tag text returns all the questions from the posts table which has this tag.

```
-- FUNCTION TO get all the questions related to a tag
create or replace function get_tag_posts(tag text)
returns SETOF posts
language plpgsql

as $$

BEGIN

 return query
 select * from posts where tags like '%'||tag||'%';
end;
$$;
```

OUTPUT:

```
select get_tag_posts ('bug');
```

Output Messages Notifications

get_tag_posts
posts

```
(1916,815,1,444,4,,1,1,<bug><status-completed><design>,"<p>With the new design, deleted answers are no longer shaded a different color.  This is confusing when trying to review/moderate the site.<
(83,815,1,199,3,,1,0,<bug><status-completed>,"<p>Related to <a href=""https://money.meta.stackexchange.com/questions/2/are-old-links-updated-automatically"">Are old links updated automatically?
(2089,1113,1,2524,2,,2,1,<discussion><support><bug>,"<p>I notice that not a single ""Generalist"" Badge has been awarded on this site. The way I understood may be incorrect. But I think quite a few w
(609,312,1,1860,4,,1,0,<bug><vote-to-close>,"<p>The tools->close page has a list of questions with ""Most Close Votes"".  This list is currently showing <em>all</em> questions in the list with exactly 1
(2274,26192,1,,5,,0,5,<support><bug><questions>,"<p>I posted a new question about 10 hours ago.  I verified it got posted and starred it as a favorite so I could reference it easier.</p>
```

l rows: 29 of 29 Query complete 00:00:00.342 Ln 426, Col 5

12) FUNCTION get_comm_of_post(p_id int) :

This function takes in the post_id and returns all the comments that post has received.

QUERY:

```
--FUNCTION TO GET_ALL COMMENTS TO A POST
create or replace function get_comm_of_post(p_id int)
returns SETOF comments
language plpgsql

as $$
DECLARE
   us_id int;c_id int;
BEGIN

 return query
 select * from comments where post_id=p_id;
end;
$$;
```

OUTPUT:

```
select get_comm_of_post(1);
```

Output Messages Notifications

get_comm_of_post comments
(1,1,854,"Possibly merged with another community? How old is the previous community?")
(2,1,149,"The previous community was about 10 months old.")
(4,1,881,"Heh I just asked the same question on meta.SO: http://meta.stackexchange.com/questions/59672/questions-on-beta-sites-are-way-older-than-they-should-be")
(15,1,3,"Yes, the site was seeded with questions from the StackExchange 1.0 site I founded, BasicallyMoney.com. User accounts appear to be intact as well, which is sw...

TRIGGERS

1)Trigger del_post:

This trigger ensures that whenever there is a delete on a post the current_user who does that is a user who created that post. That is a user can only delete a post created by him.

QUERY:

```
--CREATE TRIGGER DEL_POST: User can only delete a post created by the user.
create or replace trigger del_post
before delete
on posts
for each row
execute procedure check_del_post();

create or replace function check_del_post()
returns trigger
language plpgsql
as $$
begin
if ('user_' || old.owner_user_id :: text != current_user) then
    raise exception 'You cannot delete the posts that are not created by you';
end if;

return old;
end;
$$;
```

OUTPUT:

```
52  select owner_user_id from posts where post_id=11;
```

Data Output Messages Notifications

	owner_user_id integer
1	91

project/user_100007@PostgreSQL 16

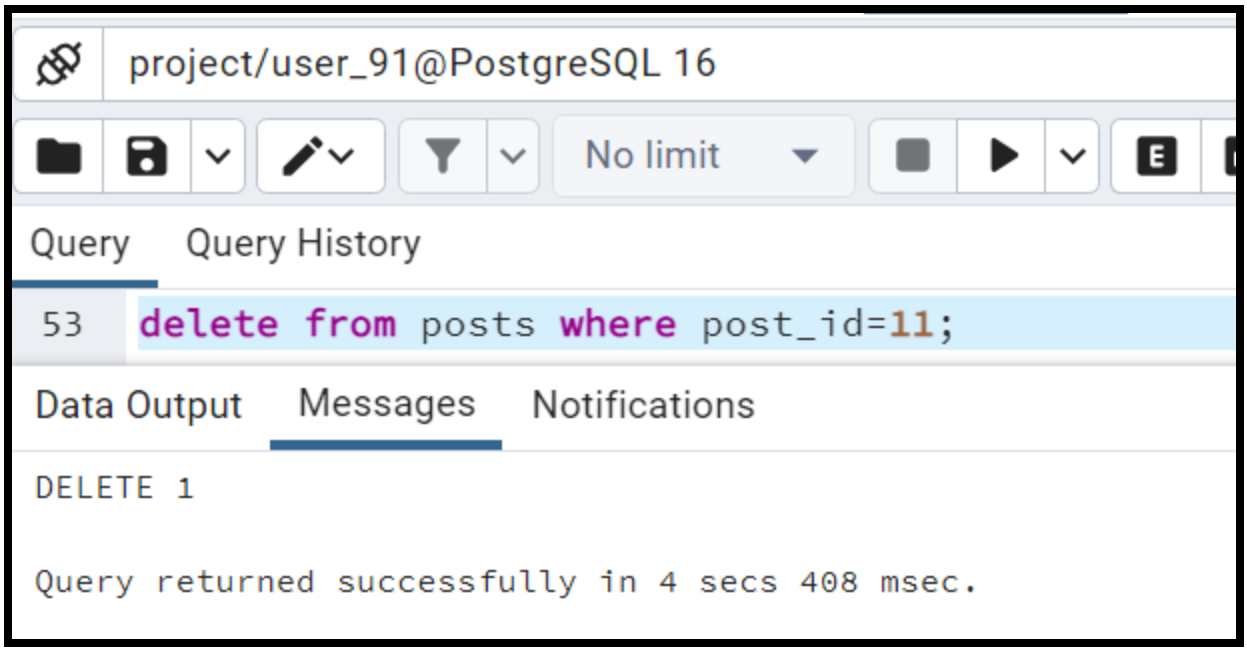
No limit

Query Query History

```
53  delete from posts where post_id=11;
```

Data Output Messages Notifications

```
ERROR:  You cannot delete the posts that are not created by you
CONTEXT:  PL/pgSQL function check_del_post() line 4 at RAISE

SQL state: P0001
```



Trigger: best_ans_upd:

This trigger ensures that whenever there is an insertion in the votes table, correspondingly the score of the post goes up or down, the score of the user whose post is getting votes goes up or down. Upvotes and downvotes are set, and due to the change of score, the best answer id is also updated.

```
--CREATE TRIGGER : UPDATE SCORE OF POST, SCORE OF USERS,BEST ANS ID OF QUESTION POST
--VIEW COUNT INCREMENTED WHEN THERE IS AN INSERTION IN VOTES
create or replace trigger best_ans_upd
after insert
on votes
for each row
execute procedure check_best_ans();
select * from posts where post_id=197;
create or replace function check_best_ans()
returns trigger
language plpgsql
as $$
DECLARE
     i int;
Begin
if (new.vote_type_id =2) then
            update  posts set score = score+1 where posts.post_id = new.post_id;
            update posts set upvotes=upvotes+1 where posts.post_id=new.post_id;
            update users set score = score + 15 where users.user_id in
            (select posts.owner_user_id from posts where posts.post_id= new.post_id limit 1);
            update users set up_votes = up_votes + 1 where users.user_id= new.user_id ;
```

```
        elsif(new.vote_type_id =3) then
            update  posts set score = score-1 where posts.post_id = new.post_id;
            update posts set downvotes=downvotes+1 where posts.post_id=new.post_id;
            update users set score = score - 8 where users.user_id in
            (select posts.owner_user_id from posts where posts.post_id= new.post_id);
            update users set down_votes = down_votes + 1 where users.user_id= new.user_id ;

end if;

if ((select posts.post_type_id from posts where posts.post_id = new.post_id)=2) then

i:=(select post_id from posts where score in (select max(score) from posts where parent_id=new.parent_id) limit 1);
update posts set best_answer_id = i  where posts.post_id = new.parent_id;
end if;
update users set users.view_count = users.view_count+1 where users.user_id in
(select owner_user_id from posts where posts.post_id =new.post_id );
return NULL;
end;
$$;
```

OUTPUT:

2092 is a question post whose initial best_answer_id is 2105, while 2099 also has an equal score

```
select post_id,score from posts where parent_id=2092;
```

Output Messages Notifications

post_id [PK] integer	score integer
2099	5
2105	5
2097	4

```
select best_answer_id from posts where post_id=2092;
```

Output Messages Notifications

best_answer_id integer
2105

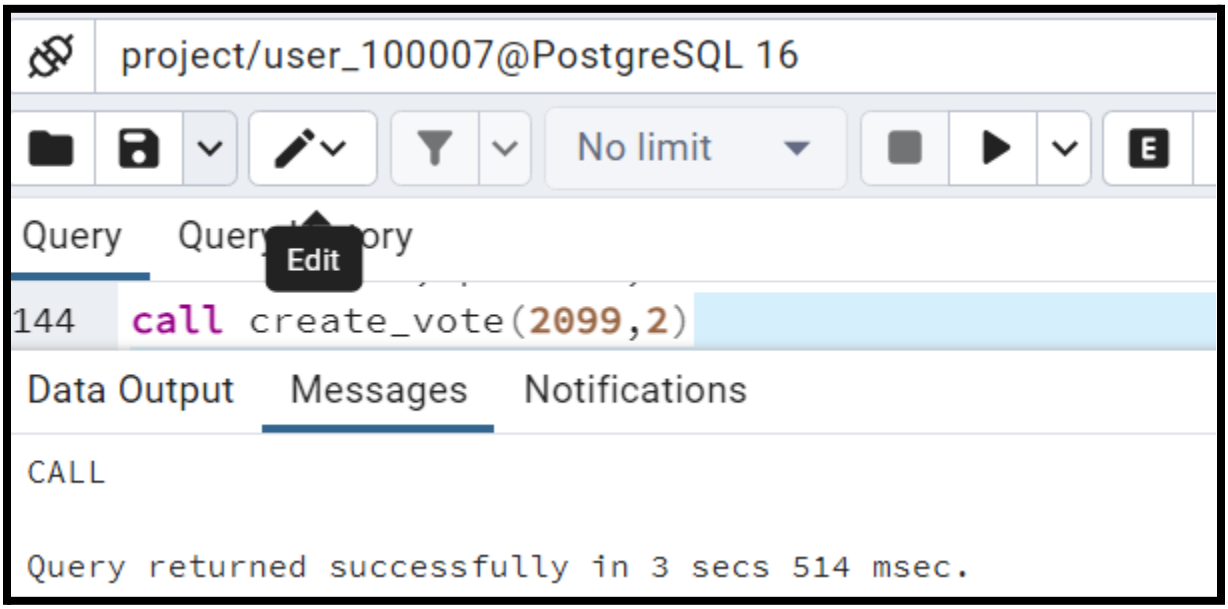
The owener_user id of 2099 had a score of 597 initially

```
select score,up_votes,down_votes from users where user_id=11604;
```

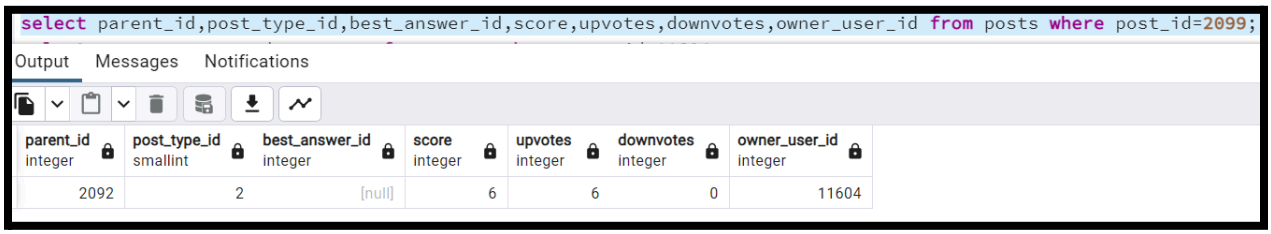
Output Messages Notifications

score integer	up_votes integer	down_votes integer
597	13	2

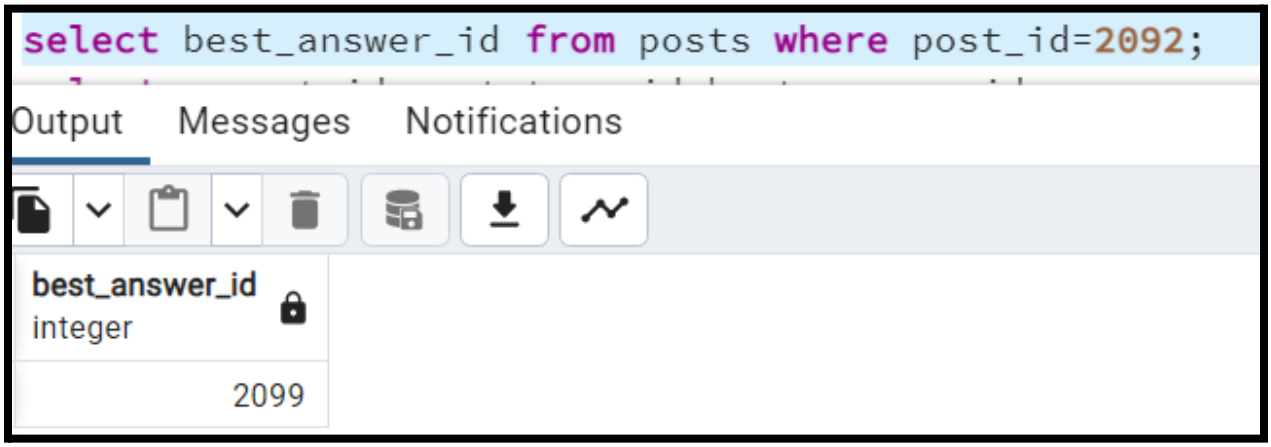
We create an upvote for the post 2099.



As a result the upvote and score count for 2099 goes up to 6.



Thus now 2099 has max score is now the best answer id for 2092.



The score of the user id who created 2099 also went up by 15 points.

```
select score,up_votes,down_votes from users where user_id=11604;
```

Output Messages Notifications

score integer	up_votes integer	down_votes integer
612	13	2

3)

Trigger upd_ans_ct:

Increment answer count of answer posts in posts table.

```
--TRIGGER TO INCREMENT ANSWER COUNT ON INSERTION OF ANSWER POST IN POSTS
create or replace trigger upd_ans_ct
after insert
on posts
for each row
execute procedure check_ans_ct();
create or replace function check_ans_ct()
returns trigger
language plpgsql
as $$
DECLARE
     i int;
Begin

 if (new.post_type_id = 2) then
    update posts set answer_count = answer_count+1 where posts.post_id = new.parent_id;
 elseif(new.post_type_id=1)then
        update posts set answer_count = answer_count where posts.post_id = new.parent_id;

    end if;
end;
$$;
```

Initially the answer count for question post 2918 was 0.

```
select post_type_id,answer_count from posts where post_id=2918;
```

Output Messages Notifications

post_type_id smallint	answer_count integer
1	0

We created an answer post which had a parent id = 2918.

```
169 call create_new_post (2,'test_answer',2918,'newtag');
```

Data Output Messages Notifications

```
NOTICE:  100007
CALL

Query returned successfully in 3 secs 257 msec.
```

Now the answer count for 2918 increased.

```
select post_type_id,answer_count from posts where post_id=2918;
```

Output Messages Notifications

post_type_id smallint	answer_count integer
1	1

4)

Trigger upd_tag_ct:

It checks if the tag to be inserted is in the tags table. If yes, it increments it, else it add it in the tags table

```
-- TRIGGER FOR TAG UPDATION.
create or replace trigger upd_tag_ct
after insert
on posts
for each row
execute procedure check_tag_ct();

create or replace function check_tag_ct()
returns trigger
language plpgsql
as $$
DECLARE
     i int; t_id int;
Begin
    t_id := (select max(tag_id) from tags)+1;
    IF NOT EXISTS (select * from tags where tag_desc = new.tags) then
        insert into tags values(t_id,new.tags,1);
    ELSE
        update tags set count=count+1 where tag_desc=new.tags;
    END IF;
    return NULL;
end;
$$;
```

Initially the count for the tag 'newtag' =0

```
select * from tags where tag_desc = 'newtag'
```

Output Messages Notifications

tag_id [PK] integer	tag_desc character varying (255)	count integer
139	newtag	0

We then created a post for, with the same tag 'newtag'

```
169  call create_new_post (2,'test_answer',2918,'newtag');
```

Data Output | Messages | Notifications

```
NOTICE:  100007
CALL

Query returned successfully in 3 secs 257 msec.
```

Now the tag count for the tag 'newtag' incremented by 1

```
select * from tags where tag_desc = 'newtag'
```

Output | Messages | Notifications

tag_id [PK] integer	tag_desc character varying (255)	count integer
139	newtag	1

5) Trigger upd_ans_ct_del :

Decrements the answer count in the posts table and the tag count in the tags table when there is a deletion in the posts table.

```
--TRIGGER TO DECREMENT ANSWER COUNT ON DELETION OF ANSWER POST AND TAG COUNT IN POSTS

create or replace trigger upd_ans_ct_del
after delete
on posts
for each row
execute procedure check_del_ans_ct();
create or replace function check_del_ans_ct()
returns trigger
language plpgsql
as $$
DECLARE
     i int;
Begin
if (old.post_type_id = 2) then
    update posts set answer_count = answer_count-1 where posts.post_id = old.parent_id;
        end if;

    update tags set count=count-1 where tags.tag_desc=old.tags;
    return NULL;
end;
$$;
```

OUTPUT

3096 is the answer post for question for 2918, which has a tag 'newtag'

```
select parent_id,tags from posts where post_id=3096;
```

Output Messages Notifications

parent_id integer	tags character varying (512)
2918	newtag

2918 has only one answer.

```
select answer_count from posts where post_id =2918;
```

Output Messages Notifications

answer_count integer
1

The tag count for new tag is also 1

```
select * from tags where tag_desc = 'newtag'
```

Output Messages Notifications

tag_id [PK] integer	tag_desc character varying (255)	count integer
139	newtag	1

We delete the post 3096.

```
delete from posts where post_id =3096
```

Data Output Messages Notifications

```
DELETE 1

Query returned successfully in 571 msec.
```

```
select answer_count from posts where post_id =2918;
```

Output Messages Notifications

answer_count integer
0

```
select * from tags where tag_desc = 'newtag'
```

Output Messages Notifications

tag_id [PK] integer	tag_desc character varying (255)	count integer
139	newtag	0

The answer_count and the tag_count for the post goes down upon its deletion.

6)

Trigger upd_comm_ct:

Increment comment count corresponding to a post when there is an insertion in the comments table.

```
--TRIGGER TO INCREMENT COMMENT COUNT ON INSERTION IN COMMENTS TABLE FOR THAT POST
create or replace trigger upd_comm_ct
after insert on comments
for each row
execute procedure check_comm_ct();

create or replace function check_comm_ct()
returns trigger
language plpgsql
as $$
DECLARE
     i int;
Begin

update posts set comment_count = comment_count+1 where posts.post_id = new.post_id;
    return NULL;
end;
$$;
```

OUTPUT:

Initially comment count for post 2918 was 1.

```
select comment_count from posts where post_id=2918;
```

Output Messages Notifications

comment_count integer
1

Then we created another comment for post 2918.

```
call create_new_comment (2918,'Black Adam');
```

Data Output | Messages | Notifications

```
CALL

Query returned successfully in 38 msec.
```

As a result the comment count increased by 1.

```
select comment_count from posts where post_id=2918;
```

Data Output | Messages | Notifications

	comment_count integer
1	2

7)Trigger : upd_del_comm_ct:

Decrements the comments count in the posts table whenever there is a deletion of a post.

```
--TRIGGER TO DECREMENT COMMENT COUNT ON DELETION IN COMMENTS TABLE FOR THAT POST
create or replace trigger upd_del_comm_ct
after delete on comments
for each row
execute procedure check_del_comm_ct();

create or replace function check_del_comm_ct()
returns trigger
language plpgsql
as $$
DECLARE
     i int;
Begin

update posts set comment_count = comment_count-1 where posts.post_id = old.post_id;
    return NULL;
end;
$$;
```

OUTPUT:

Initially the comment count for post 2918 =2.

```
443  select comment_count from posts where post_id=2918;
```

Data Output Messages Notifications

	comment_count integer
1	2

Now we delete the comment.

```
delete from comments where comment_text = 'Black Adam'
```

Data Output | Messages | Notifications

```
DELETE 1

Query returned successfully in 64 msec.
```

Then the comment count decreases by 1.

```
select comment_count from posts where post_id=2918;
```

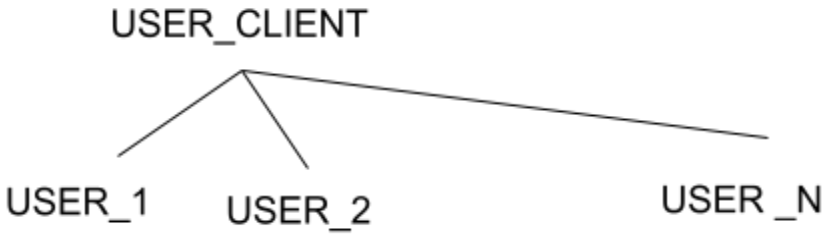
Output | Messages | Notifications

comment_count integer
1

All Roles and list of privileges given to them

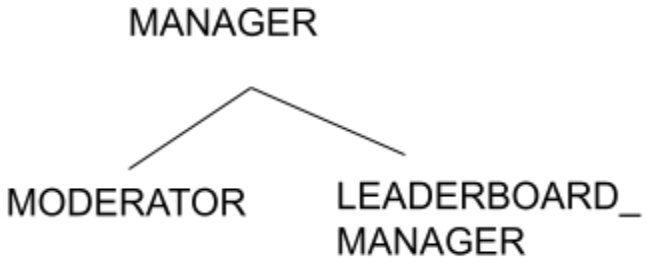
Group Role : User_client

All users are defined under the user_client. For the existing data, i.e for all the existing user_ids , a **user role** has been created under the user_client with username : user_<user_id>.



Group Role : Managers

Two roles have been created under managers: moderator and leaderboard_manager



CLIENT_USER

	SELECT	INSERT	DELETE	UPDATE
USERS	✓	✓	✓	✓
POSTS	✓	✓	✓	
BADGES	✓			
TAGS	✓			
COMMENTS	✓	✓	✓	
VOTES		✓		

MANAGER

	SELECT	INSERT	DELETE	UPDATE
USERS	✓		✓	✓
POSTS	✓		✓	✓
BADGES	✓			✓

TAGS	✓			
COMMENTS	✓			
VOTES	✓			

VIEWS

1) VIEW: leader_board

This view stores the details of all users whose ranking in the leaderboard is from 1 to 50, i.e the top 50 highest scorers. In a social media , reward based platform like this one , this is commonly checked out by the competitive class of users , to see if they are among the top 50 , or if their ranking has dropped or gone up

```
--VIEW 1: TOP 50 IN LEADERBOARD
create or replace view leader_board
as
select users.user_id,users.user_name,users.score,badges.class,users.user_bio
from users, badges where users.user_id = badges.user_id order by users.score desc limit 50;

select * from leader_board
select * from users
```

```
create or replace view leader_board
as
select users.user_id,users.user_name,users.score,badges.class,users.user_bio
from users, badges where users.user_id = badges.user_id order by users.score desc limit 50;

select * from leader_board
```

Data Output Messages Notifications

	user_id integer	user_name character varying (255)	score integer	class smallint	user_bio text
1	187	JTP - Apologise to Monica	10680	1	Father, husband, puppy person, and financial author.
2	366	GS - Apologise to Monica	10259	1	Monica Cellio has been unfairly treated. <a href="https://meta.stackexchange.com/questions/33652
3	10997	Ben Miller - Remember Monica	6542	1	Debt avenger, YNAB user, HSA aficionado
4	3	Chris W. Rea	4933	1	Husband, dad, computer geek, independent software developer, Canadian.
5	91	MrChrister	4706	1	I have a basic understanding about money. I try to stay up to date on personal finance and as a web
6	1113	Dheer	3928	1	I moved from being a developer and am currently a business analyst in the banking domain.
7	815	Alex B	2451	1	[null]
8	2998	littleadv	2439	1	I'm mostly on Money@SE, although I do occasionally stop by at StackOverflow to play with the little
9	5414	mhoran_psprep	1922	1	[null]
10	12894	Joe	1901	1	SAS Programmer/Developer/Analyst/buzzword, when I'm not parenting a pair of rambunctious little

rows: 50 of 50 Query complete 00:00:00.095 Ln 654, Col 67

VIEW 2: featured_questions :

This view stores the details of top 15 highly rated question posts. This is to cater to the newly joined curious users who check out the questions asked and liked by the majority.

```
--View 2: top 15 HIGH SCORED QUESTIONS
create or replace view featured_questions
as
select posts.post_id,posts.owner_user_id , posts.body, posts.score, posts.tags
from posts where posts.post_type_id=1 order by posts.score desc limit 15;
select * from featured_questions
```

Data Output Messages Notifications

	post_id integer	owner_user_id integer	body text
1	2943	103859	<p>I need to file taxes soon for my business. I'm reading some accounting textbooks to teach myself the subject. I want to ask questions about it. I
2	2716	213	<p><a href="https://money.stackexchange.com/questions/109286/should-i-replace-some-parts-of-my-portfolio-with-an-option-strategy/">Should I r
3	2894	20480	<p>Tags <a href="https://money.stackexchange.com/questions/tagged/volume" class="post-tag" title="show questions tagged 'volume'"
4	2864	213	<p>GS didn't close this post that just quoted external source, but did recently close new posts that did. Did the policy change? Is this a contradictior

rows: 15 of 15 Query complete 00:00:00.468 Ln 665, Col 1

VIEW 3: top5_tags :

This view contains the tags with the most number of posts. That is the highly used 5 tags.

This will give an idea to the users about the type of questions discussed in the forum, and becomes a trademark for the platform for those kind of questions.

```
--Top Tags with question count
create or replace view top5_tags
as
select tag_desc,count from tags order by count desc limit 5;
select * from top5_tags;
```

Data Output Messages Notifications

	tag_desc character varying (255)	count integer
1	discussion	580
2	tags	88
3	allowed-topics	73
4	status-completed	70
5	support	55

ows: 5 of 5 Query complete 00:00:00.119

INDICES

```
create index user_id_index on users using hash(user_id);
create index user_id_index2 on posts using hash(post_id);
create index user_id_index3 on votes using hash(vote_id);
create index user_index4 on users (score);
```

We are using hashing on users(user_id) , posts(post_id) and votes(vote_id). The functions and procedures mentioned above are the most frequently used queries by a user or manager.

Functions that use has index on user_id : get_user_info() ,ban_users()

Functions that use hash index on post_id : delete_post(),get_ans_of_post()

Functions that use hash index on vote_id : Triggers: best_ans_upd - This trigger is activated whenever there is a vote insertion, which is very frequent.

Functions that use btree index on score: ban_users() - This is a frequent query used for regular updation by the moderator.

Given the equality constraints applied to these fields in queries, opting for a hash map index stands out as the most fitting choice. Hashing is an effective technique to calculate the direct location of a data record on the disk without using index structure. By leveraging hash functions with search keys as inputs, hashing efficiently generates data record addresses, surpassing the speed of traditional B-tree methods.

QUERIES

Query 1:

Create a procedure which takes in a post id and inserts a comment 'you have score <3' if it the post has a score less than 3

```
create or replace procedure query_1(p_id int)
language plpgsql
as $$
DECLARE
    role_name TEXT; tt text;
Begin
        if((select score from posts where post_id=p_id)<3)then
            call create_new_comment(p_id,'you have score<3');
        end if;
       end;
$$;
```

```
call query_1(1);
```

Data Output Messages Notifications

	comment_text text
1	Possibly merged with another community? How old is the previous community?
2	The previous community was about 10 months old.
3	Heh I just asked the same question on meta.SO: http://meta.stackexchange.com/questions/59672/questions-on-beta-sites-are-way-older-than-they-should-be
4	Yes, the site was seeded with questions from the StackExchange 1.0 site I founded, BasicallyMoney.com. User accounts appear to be intact as well, which is s…
5	you have score<3

Query 2: Update a user to have score less than -50 and Switch to moderators role and run his functionalities, and revert back.

```
update users set score =-60 where user_id=121939
```

Data Output | Messages | Notifications

```
UPDATE 1

Query returned successfully in 47 msec.
```

project/moderator@PostgreSQL 16

```
call ban_users();
select user_bio from users where user_id=121939;
```

Data Output | Messages | Notifications

	user_bio text
1	user_id_to_be_deleted

Query 3: update the score of one user to 5000 and change to leaderboard manager and show the changes, and then revert back to its original score.

```
select * from badges where user_id=213;
call update_badge_class();
select * from badges where user_id=213;
```

Data Output Messages Notifications

	badge_id [PK] integer	user_id integer	class smallint
1	17	213	3

```
update users set score=5000 where user_id=213;

select * from users where user_id=213;
call update_badge_class();
```

Data Output Messages Notifications

	user_id [PK] integer	user_name character varying (255)	score integer	vie int
1	213	Jim G.	5000	

project/leaderboard_manager@PostgreSQL 16

```
call update_badge_class();
select * from badges where user_id=213;
```

Data Output Messages Notifications

	badge_id [PK] integer	user_id integer	class smallint
1	17	213	1

Query 4: for a given user, delete all the posts posted by him with a tag 'bug'

project/user_2998@PostgreSQL 16

```
select * from posts where owner_user_id=2998 and tags like '%bug%';
```

Data Output Messages Notifications

	post_id [PK] integer	owner_user_id integer	post_type_id smallint	best_answer_id integer	score integer	parent_id integer	answer_count integer
1	1924	2998	1	1929	2	[null]	
2	490	2998	1	491	4	[null]	

```
DO $$
DECLARE
    i INTEGER;
begin
for i in (select post_id from posts where tags like '%bug%' and owner_user_id =
          (SELECT SUBSTRING(current_user FROM POSITION('_' IN current_user) + 1))::int)loop
          call delete_post(i);
        end loop;
end;$$;
```

```
select * from posts where owner_user_id=2998 and tags like '%bug%';
```

Data Output Messages Notifications

	post_id [PK] integer	owner_user_id integer	post_type_id smallint	best_answer_id integer	score integer	parent_id integer

Query 5: for a given tag, create a comment for all the post which has that tag , and show the changes in the comment_count.

```
select * from posts where tags like '%meta%'
```

Data Output Messages Notifications

	post_id [PK] integer	owner_user_id integer	post_type_id smallint	best_answer_id integer	score integer	parent_id integer	answer_count integer	comment_count integer
1	168	1113	1	169	2	[null]	2	2
2	2731	87194	1	2732	5	[null]	1	0

```
DO $$
DECLARE
    i INTEGER;
begin
for i in (select post_id from posts where tags like '%meta%')loop
          call create_new_comment(i,'this is a meta post');
        end loop;
end;$$;
```

```
select * from posts where tags like '%meta%'
```

Data Output Messages Notifications

	post_id [PK] integer	owner_user_id integer	post_type_id smallint	best_answer_id integer	score integer	parent_id integer	answer_count integer	comment_count integer
1	168	1113	1	169	2	[null]	2	3
2	2731	87194	1	2732	5	[null]	1	1