

Data Management Homework 1 and Homework 2

Stats StackExchange Network

[Dataset](#)

[Schema](#)

[Queries](#)

Stats StackExchange Network


Gian Alvin Guico - 2033024

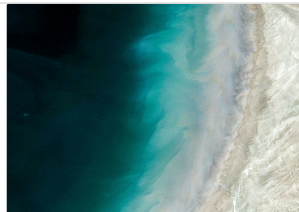
Simone Fiorellino - 1960415

Stack Exchange

Stack Exchange Q&A communities are different.

Here's how: Expert

 <https://stackoverflow.com/>



Dataset

StackExchange is a network of question-and-answer (Q&A) websites on topics in different fields, where users, questions, and answers are subject to a reputation award process.

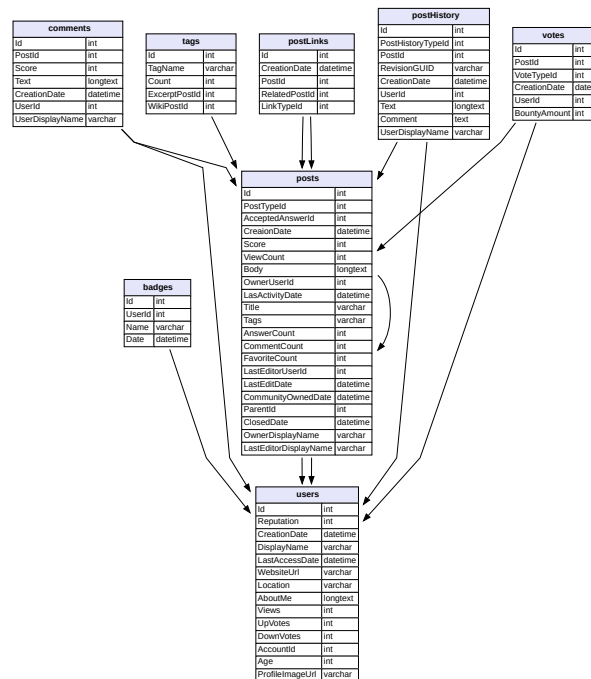
Schema

As we can see, our database is structured into eight tables

Queries

The aim of our queries is to retrieve information about certain users or posts

1. Return the average of links in a post in the top 100 most viewed posts



```
# Duration: 0.766s
select round(avg(count_id),2)
from (select count(pl.Id) as count_id
from postLinks pl, posts p
where p.Id=pl.PostId
group by pl.PostId
order by ViewCount desc
limit 100) as cl;
```

- The average of the links present in one post of the most viewed post are slightly greater than the global average.

$$\text{global_avg} = 1.46 < \text{top_avg} = 1.78$$

This result make sense.

2. Return the most viewed posts of the user with max(reputation) ordered by the "ViewCount".

```
# Duration: 0.297
select u.DisplayName, p.Title, p.ViewCount, u.Reputation
from posts p, users u
where u.Reputation = (select max(u.Reputation) from users u) and u.Id=p.OwnerUserId and p.Title is not null
order by p.ViewCount desc;
```

4. For every user, return the DisplayName and his last post

```
# Duration: 0.172s
select u.Id, max(p.CreationDate), u.DisplayName
from posts p join users u on p.OwnerUserId = u.Id
group by u.Id;
```

5. For each user return, in the "history", how many times, he/she had talked about arithmetic topic

```
# Duration: 1.7
select u.Id, u.DisplayName, count(u.Id) as times, p.Title
from postHistory ph, users u, posts p, tags t
where t.TagName = "arithmetic" and p.Title is not null
and u.Id = ph.UserId and p.Id = ph.PostId
group by u.Id, p.Title
order by times desc
```

6. Return the post with most interactions with "math" in the Title

```
# Duration: 0.403s
select p.Id, p.Title, (p.AnswerCount + p.CommentCount) as sum
from posts p
where p.AnswerCount is not null and p.CommentCount is not null
and p.Title like "%math%"
order by sum desc;
```

```
#####
#####

# creating a new table to avoid some operation
CREATE TABLE posts_opt(
  Id int,
  Title varchar(255),
  Interaction int,
  PRIMARY KEY(Id));

INSERT INTO posts_opt(
  SELECT Id, Title, (AnswerCount + CommentCount)
  FROM posts
  where AnswerCount is not null and CommentCount is not null);

# Duration: 0.047
select p.Id, p.Title, p.Interaction
from posts_opt p
where p.Title like "%math%"
order by p.Interaction desc;
```

- Return the user id, user name, and the count of the badges “Popular Question” of the users that obtained at least one badge “Popular Question” and have a reputation greater than the average.

```
# Duration: 0.110s
select u.Id, u.DisplayName, count(distinct b.Id) as Badges
from users u
join badges b on b.UserId = u.Id
where b.Name = 'Popular Question' and u.Reputation >
(select avg(Reputation)
 from users)
group by u.Id
order by Badges desc;
```

If we compute the mean of these result:

```
# Duration: 0.141s
select avg(bb.badges)
from (select count(distinct b.Id) as badges
from users u
join badges b on b.UserId = u.Id
where b.Name = 'Popular Question' and u.Reputation >
(select avg(Reputation)
 from users)
group by u.Id) as bb;
```

We can observe that the distribution of the “badges for user” follow a power-law distribution, i.e. just a few users have a lot of “Popular Question”.

- Users “beta tester” with an age greater than 40 that have commented. Given these users we want to know their locations. How many users we have in these locations?

```
# Users “beta tester” with an age greater than 40 that have commented
select distinct b.UserId, u.DisplayName
from badges b join comments c on c.UserId = b.UserId
```

```

join users u on u.Id = b.UserId
where b.Name like "Beta%" and u.Age > 40;

# given these users we want to know their locations
select u.Location
from users u join
(select distinct b.UserId, u.DisplayName
from badges b join comments c on c.UserId = b.UserId
join users u on u.Id = b.UserId
where b.Name like "Beta%" and u.Age > 40) as sub_set on u.Id=sub_set.UserId
where u.Location is not null
group by u.Location;

# Duration: 0.078s
# How many users we have in these locations?
select sum(users_for_country)
from (
  select count(u.Id) as users_for_country
  from users u join (
    select u.Location
    from users u join (
      select distinct b.UserId, u.DisplayName
      from badges b join comments c on c.UserId = b.UserId
      join users u on u.Id = b.UserId
      where b.Name like "Beta%" and u.Age > 50
    ) as sub_set on u.Id=sub_set.UserId
    where u.Location is not null
    group by u.Location
  ) as ss1 on ss1.Location=u.Location
  group by u.Location
) as final;

```

8. Count the number of badges of the users with positive sentiment, e.g. `u.UpVotes - u.DownVotes`

```

# Duration: 0.313
select distinct u.DisplayName, (u.UpVotes - u.DownVotes) as sentiment, count(b.Id)
from users u join badges b on b.UserId = u.Id
group by u.Id
having sentiment > 0
order by sentiment desc;

#####
#####

alter table users
add Sentiment int;

update users u
set u.Sentiment = (u.UpVotes - u.DownVotes);

# Duration: 0.1
select distinct u.DisplayName, u.Sentiment, u.badges_count
from users u
group by u.Id
having sentiment > 0
order by sentiment desc;

```

- Given this result, we observed that the users with better sentiment have more badges with respect to those with a lower sentiment score.

9. Avg age of the “beta tester” users who have commented, these users have a positive “sentiment”.

```
# Duration: 0.625
select avg(u.Age)
from badges b
join comments c on c.UserId = b.UserId
join users u on u.Id = b.UserId
where u.DisplayName in (select distinct u.DisplayName
                        from users u join badges b on b.UserId = u.Id
                        where b.Name like "%beta%" and (u.UpVotes - u.DownVotes) > 0
                        group by u.DisplayName)
and u.DisplayName is not null
```

10. How many badges have the most popular StackExchange users in March 2013, in terms of comments and answers

```
# Duration: 6.200s
select distinct p.ParentId, u.DisplayName, b1.badges_count
from (select distinct p.ParentId
from posts p
where p.Id is not null and p.ParentId is not null and
      p.CreationDate>"2013-03-00 00:00:00" and p.CreationDate<"2013-04-00 00:00:00"
order by p.CommentCount desc
limit 5) as p1,
(select distinct p.ParentId
from posts p
where p.Id is not null and p.ParentId is not null and
      p.CreationDate>"2013-03-00 00:00:00" and p.CreationDate<"2013-04-00 00:00:00"
order by p.AnswerCount desc
limit 5) as p2,
posts as p
join users u on p.ParentId = u.Id
join (select b.UserId, count(Id) as badges_count
from badges b
group by b.UserId) as b1 on b1.UserId = p.ParentId
where p1.ParentId = p.ParentId or p2.ParentId=p.ParentId
group by p.ParentId;

#####
#####

# Same query but we have used the view
create view Top5Comment as
select distinct p.ParentId
from posts p
where p.Id is not null and p.ParentId is not null and
      p.CreationDate>"2013-03-00 00:00:00" and p.CreationDate<"2013-04-00 00:00:00"
order by p.CommentCount desc
limit 5;

create view Top5Answer as
select distinct p.ParentId
from posts p
where p.Id is not null and p.ParentId is not null and
      p.CreationDate>"2013-03-00 00:00:00" and p.CreationDate<"2013-04-00 00:00:00"
order by p.AnswerCount desc
limit 5;

create view count_budget_for_user as
select b.UserId, count(Id) as badges_count
```

```

from badges b
group by b.UserId;

create view Top5CA as
select *
from Top5Comment
union
select *
from Top5Answer;

select distinct t.ParentId, u.DisplayName, cb.badges_count
from users u, Top5CA t, count_budget_for_user cb
where u.Id=t.ParentId and cb.UserId=t.ParentId;

#####
#####

# create a new column for users
alter table users
add badges_count int;

# fill the column with the values found in the query.
update users u
set u.badges_count =
(select cb.badges_count
from count_budget_for_user cb
where u.Id = cb.UserId)
where exists
(select cb.badges_count
from count_budget_for_user cb
where u.Id = cb.UserId);

# Duration: 6.000
select distinct t.ParentId, u.DisplayName, u.badges_count
from users u, Top5CA t
where u.Id=t.ParentId;

```

- With this variation of the table users, the query speeds up by 0.2s

11. return title, the body of the post, the count of comments, the count of the users that have modified the post, and the count of modifying. The post has to have in the “body” the word “proof” and in the tags the tag “bayesian”.

```

# Duration: 0.860s
select p.Title, p.Body, count(distinct c.Id) as comment_count,
       count(distinct u.Id) users_count, count(distinct ph.Id) mod_count
from posts p
join postHistory ph on p.Id=ph.PostId
join users u on ph.UserId=u.Id
join comments c on c.PostId = p.Id
where p.Body like "%proof%" and p.Tags like "%bayesian%"
group by p.Id;

```

12. Return the users that are owners at least of one post, with a reputation bigger than 1000, and have commented on at least one post with a score bigger than 10.

```

# Duration: 6.594
select distinct u.Id, u.DisplayName
from users u
inner join posts p on u.DisplayName=p.OwnerDisplayName
where u.Reputation > 1000 and u.Id in (
    select u.Id
    from users u join comments c on u.Id=c.UserId
    where c.Score >10
);

#####
#####

# Duration: 0.219
select distinct u.Id, u.DisplayName
from users u
inner join posts p on u.Id=p.OwnerUserId
where u.Reputation > 1000 and u.Id in (
    select u.Id
    from users u join comments c on u.Id=c.UserId
    where c.Score >10
);

```

Summary

We have seen some queries on the StackExchange dataset. We saw examples of how to have optimized queries, i.e.:

- Creating new tables
- Modifying the already existing tables
- Working on primary keys

Also the views as been used, this in order to have simple, secure and reusable queries.

P.S.: Indexes was not used because the in-built optimization of the used software and the less importance of these in queries on small tables or big tables where report queries process most or all of the rows.