Database Design

Database Implementation

Database Connection on GCP:

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to cs411-pt1-stage3-402920.
Use “gcloud config set project [PROJECT_ID]” to change to a different project.
gcloud sql connect cs411-db-s3 --user=root --quietdhruvbhoj@cloudshell:~ (cs411-pt1-stage3-402920)$ gcloud sql connect cs411-db-s3 --user=root
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 100749
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| CourseOverflow     |
| classicmodels      |
| courseoverflow     |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
7 rows in set (0.00 sec)

mysql> use courseoverflow
```

```
mysql> use courseoverflow
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+--------------------------+
| Tables_in_courseoverflow |
+--------------------------+
| Subject                  |
| answers                  |
| comments                 |
| questions                |
| tags                     |
| users                    |
+--------------------------+
6 rows in set (0.00 sec)
```

DDL Commands:

```
CREATE TABLE answers(
      a_id            VARCHAR(10) NOT NULL PRIMARY KEY,
      a_body          VARCHAR(60000) NOT NULL,
      a_creation_date DATE  NOT NULL,
      q_id            VARCHAR(10) NOT NULL,
      u_id            VARCHAR(10) NOT NULL,
      FOREIGN KEY(q_id) References questions(q_id) on delete cascade on
      update cascade,
      FOREIGN KEY(u_id) References users(u_id) on delete cascade on update
      cascade
);

CREATE TABLE comments(
      c_id            VARCHAR(10) NOT NULL PRIMARY KEY,
      text            VARCHAR(5000) NOT NULL,
      c_creation_date DATE  NOT NULL,
      q_id            VARCHAR(10) NOT NULL,
      u_id            VARCHAR(10) NOT NULL,
      FOREIGN KEY(q_id) References questions(q_id) on delete cascade on
      update cascade,
      FOREIGN KEY(u_id) References users(u_id) on delete cascade on update
      cascade
);

CREATE TABLE tags(
      t_id   VARCHAR(10) NOT NULL,
      t_name VARCHAR(255),
      count  INTEGER  NOT NULL,
      PRIMARY KEY(t_id,t_name)
);

CREATE TABLE questions(
      q_id            VARCHAR(10) NOT NULL PRIMARY KEY,
      title           VARCHAR(255) NOT NULL,
      q_body          VARCHAR(60000) NOT NULL,
      q_creation_date DATE  NOT NULL,
      u_id            VARCHAR(10) NOT NULL,
      t_name          VARCHAR(255) NOT NULL,
      FOREIGN KEY (u_id) References users(u_id) on delete cascade on
      update cascade
);

CREATE TABLE users(
      u_id       VARCHAR(10) NOT NULL PRIMARY KEY,
      u_name     VARCHAR(255),
      reputation INTEGER  NOT NULL
```

```
);

CREATE TABLE Subject(
      q_id        VARCHAR(10),
      t_id        VARCHAR(10),
      t_name      VARCHAR(255),
      PRIMARY KEY (q_id, t_id, t_name),
      FOREIGN KEY (q_id) REFERENCES questions(q_id),
      FOREIGN KEY (t_id, t_name) REFERENCES tags(t_id, t_name)
);
```

Count Query for each table:

```
mysql> select count(*) from answers;
+----------+
| count(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*) from questions;
+----------+
| count(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.02 sec)

mysql> select count(*) from tags;
+----------+
| count(*) |
+----------+
|    63653 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*) from users;
+----------+
| count(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from comments;
+----------+
| count(*) |
+----------+
|     2674 |
+----------+
1 row in set (0.00 sec)
```

Advanced Queries

Questions with most comments and answers (most popular questions):

```
SELECT
        q.q_id,
        q.title,
        count(distinct c.c_id) as num_comments,
        count(distinct a.a_id) as num_answers ,
        (count(distinct c.c_id)+count(distinct a.a_id)) as total_responses
FROM questions q
LEFT JOIN comments c using (q_id)
LEFT JOIN answers a using (q_id)
GROUP BY q.q_id, q.title
ORDER BY total_responses DESC
LIMIT 15;
```

```
mysql> select q.q_id, q.title, count(distinct c.c_id) as num_comments, count(distinct a.a_id) as num_answers , (count(distinct c.c_id)+count(distinct a.a_id)) as total_responses from questions q left join comments
c using(q_id) left join answers a using(q_id) group by q.q_id, q.title order by total_responses desc limit 15;
+--------+-----------------------------------------------------------------------------------------------------+--------------+-------------+-----------------+
| q_id   | title                                                                                               | num_comments | num_answers | total_responses |
+--------+-----------------------------------------------------------------------------------------------------+--------------+-------------+-----------------+
| 98606  | Favorite Visual Studio keyboard shortcuts                                                           |            1 |          25 |              26 |
| 157354 | Is mathematics necessary for programming?                                                           |            1 |          22 |              23 |
| 143429 | What's the least useful comment you've ever seen?                                                   |            0 |          21 |              21 |
| 144734 | When is it good (if ever) to scrap production code and start over?                                  |            3 |          17 |              20 |
| 139228 | What do you do with a developer who does not test his code?                                         |            1 |          17 |              18 |
| 84340  | Why learn Perl, Python, Ruby if the company is using C++, C# or Java as the application language?   |            0 |          17 |              17 |
| 103059 | Where to start with source-control                                                                  |            0 |          17 |              17 |
| 194484 | What's the strangest corner case you've seen in C# or .NET?                                         |           17 |           0 |              17 |
| 115369 | Do you use source control for your database items?                                                  |            0 |          16 |              16 |
| 105838 | Real-world examples of recursion                                                                    |            2 |          14 |              16 |
| 111933 | Why shouldn't I use "Hungarian Notation"?                                                           |            1 |          14 |              15 |
| 145508 | Do you think a software company should impose developers a coding-style?                            |            0 |          15 |              15 |
| 145951 | What is the first thing you do when you install Visual Studio?                                      |            1 |          14 |              15 |
| 139214 | Redundant code constructs                                                                           |            1 |          13 |              14 |
| 109997 | How do you protect your software from illegal distribution?                                         |            5 |           9 |              14 |
+--------+-----------------------------------------------------------------------------------------------------+--------------+-------------+-----------------+
15 rows in set (0.09 sec)
```

Top Answers based to questions based on User Reputation Points :

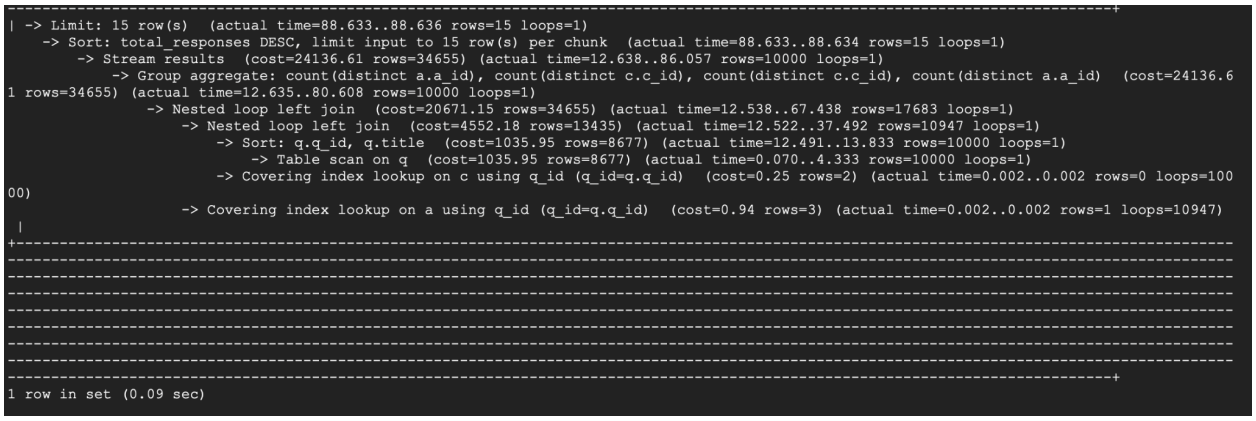
Answer body content has not been selected and shown, as the answers are comparatively long to be shown in screenshots.

```
SELECT
        a.a_id,
        a.a_creation_date,
        a.q_id,
        a.u_id,
        u.reputation
FROM
        answers a
JOIN users u ON a.u_id = u.u_id
WHERE
        (a.q_id, u.reputation) IN (
        SELECT
                a.q_id,
                MAX(u.reputation) as max_reputation
        FROM
                answers a
        JOIN users u ON a.u_id = u.u_id
        GROUP BY
                a.q_id)
ORDER BY
        a.q_id,
        a.a_creation_date ASC
LIMIT 15;
```

```
+--------+-----------------+--------+-------+------------+
| a_id   | a_creation_date | q_id   | u_id  | reputation |
+--------+-----------------+--------+-------+------------+
| 100891 | 2008-09-19      | 100001 | 11421 |       5715 |
| 100075 | 2008-09-19      | 100007 | 14954 |       1359 |
| 100458 | 2008-09-19      | 100038 | 18219 |       2810 |
| 138347 | 2008-09-26      | 100038 | 18219 |       2810 |
| 100198 | 2008-09-19      | 100045 | 13552 |     129819 |
| 100111 | 2008-09-19      | 100053 | 15127 |       7939 |
| 100159 | 2008-09-19      | 100081 | 18575 |      11552 |
| 103790 | 2008-09-19      | 100089 | 13447 |      45769 |
| 100131 | 2008-09-19      | 100107 | 15627 |       2057 |
| 100138 | 2008-09-19      | 100123 | 17613 |       3110 |
| 100271 | 2008-09-19      | 100187 | 17516 |     184366 |
| 100638 | 2008-09-19      | 100216 | 13238 |       7515 |
| 100589 | 2008-09-19      | 100228 | 10991 |       1710 |
| 100682 | 2008-09-19      | 100235 | 15541 |     112912 |
| 100251 | 2008-09-19      | 100242 | 17134 |         25 |
+--------+-----------------+--------+-------+------------+
15 rows in set (0.04 sec)
```

INDEXING

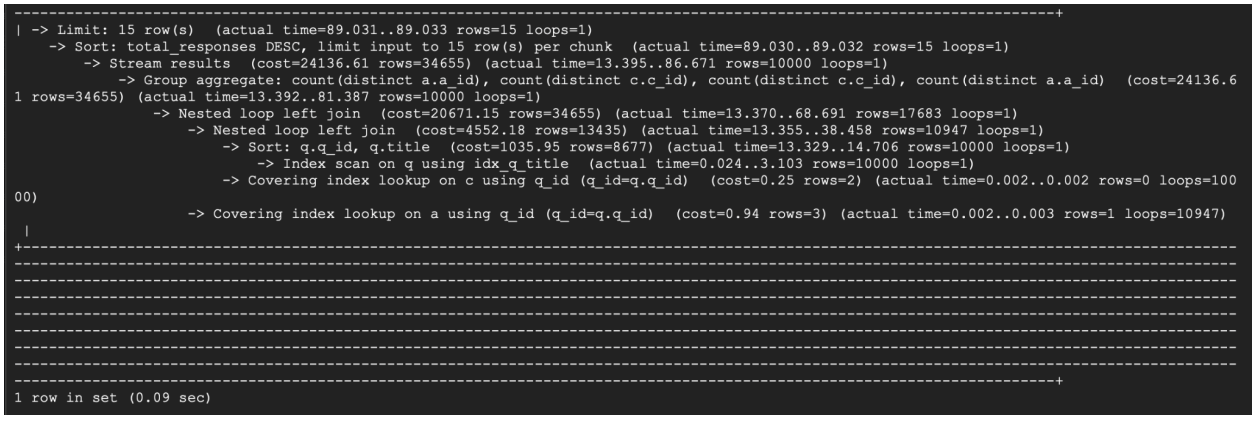
1. Running explain analyse on the **first** advanced query without indexing



```
| -> Limit: 15 row(s)  (actual time=88.633..88.636 rows=15 loops=1)
    -> Sort: total_responses DESC, limit input to 15 row(s) per chunk  (actual time=88.633..88.634 rows=15 loops=1)
        -> Stream results  (cost=24136.61 rows=34655) (actual time=12.638..86.057 rows=10000 loops=1)
            -> Group aggregate: count(distinct a.a_id), count(distinct c.c_id), count(distinct c.c_id), count(distinct a.a_id)  (cost=24136.6
1 rows=34655) (actual time=12.635..80.608 rows=10000 loops=1)
                -> Nested loop left join  (cost=20671.15 rows=34655) (actual time=12.538..67.438 rows=17683 loops=1)
                    -> Nested loop left join  (cost=4552.18 rows=13435) (actual time=12.522..37.492 rows=10947 loops=1)
                        -> Sort: q.q_id, q.title  (cost=1035.95 rows=8677) (actual time=12.491..13.833 rows=10000 loops=1)
                            -> Table scan on q  (cost=1035.95 rows=8677) (actual time=0.070..4.333 rows=10000 loops=1)
                        -> Covering index lookup on c using q_id (q_id=q.q_id)  (cost=0.25 rows=2) (actual time=0.002..0.002 rows=0 loops=100
00)
                    -> Covering index lookup on a using q_id (q_id=q.q_id)  (cost=0.94 rows=3) (actual time=0.002..0.002 rows=1 loops=10947)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.09 sec)
```

1. Indexed using the following query. I chose to index on the question title because the advanced query was grouping by question title. As you can see below it did not improve the runtime. We feel this is because performance bottlenecks likely occurs in other parts of the query.

```
mysql> create index idx_q_title on questions (title);
```



```
| -> Limit: 15 row(s)  (actual time=89.031..89.033 rows=15 loops=1)
    -> Sort: total_responses DESC, limit input to 15 row(s) per chunk  (actual time=89.030..89.032 rows=15 loops=1)
        -> Stream results  (cost=24136.61 rows=34655) (actual time=13.395..86.671 rows=10000 loops=1)
            -> Group aggregate: count(distinct a.a_id), count(distinct c.c_id), count(distinct c.c_id), count(distinct a.a_id)  (cost=24136.6
1 rows=34655) (actual time=13.392..81.387 rows=10000 loops=1)
                -> Nested loop left join  (cost=20671.15 rows=34655) (actual time=13.370..68.691 rows=17683 loops=1)
                    -> Nested loop left join  (cost=4552.18 rows=13435) (actual time=13.355..38.458 rows=10947 loops=1)
                        -> Sort: q.q_id, q.title  (cost=1035.95 rows=8677) (actual time=13.329..14.706 rows=10000 loops=1)
                            -> Index scan on q using idx_q_title  (actual time=0.024..3.103 rows=10000 loops=1)
                        -> Covering index lookup on c using q_id (q_id=q.q_id)  (cost=0.25 rows=2) (actual time=0.002..0.002 rows=0 loops=100
00)
                    -> Covering index lookup on a using q_id (q_id=q.q_id)  (cost=0.94 rows=3) (actual time=0.002..0.003 rows=1 loops=10947)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.09 sec)
```

2. Indexed using the following query. I chose to index on the question id of answers as the question id is being used to performs joins and having the answers indexed on question id might help with the runtime. As you can see below it did not improve the runtime and we feel this is because question id was a foreign key so it was already indexed and cannot be improved more.

```
mysql> create index idx_q_id on answers (q_id);
```

```
| -> Limit: 15 row(s)  (actual time=89.436..89.438 rows=15 loops=1)
    -> Sort: total_responses DESC, limit input to 15 row(s) per chunk  (actual time=89.435..89.436 rows=15 loops=1)
        -> Stream results  (cost=24136.61 rows=34655) (actual time=14.540..86.995 rows=10000 loops=1)
            -> Group aggregate: count(distinct a.a_id), count(distinct c.c_id), count(distinct c.c_id), count(distinct a.a_id)  (cost=24136.6
1 rows=34655) (actual time=14.537..81.936 rows=10000 loops=1)
                -> Nested loop left join  (cost=20671.15 rows=34655) (actual time=14.512..69.217 rows=17683 loops=1)
                    -> Nested loop left join  (cost=4552.18 rows=13435) (actual time=14.489..39.339 rows=10947 loops=1)
                        -> Sort: q.q_id, q.title  (cost=1035.95 rows=8677) (actual time=14.456..15.755 rows=10000 loops=1)
                            -> Index scan on q using idx_q_title  (actual time=0.025..3.037 rows=10000 loops=1)
                        -> Covering index lookup on c using q_id (q_id=q.q_id)  (cost=0.25 rows=2) (actual time=0.002..0.002 rows=0 loops=100
00)
                    -> Covering index lookup on a using q_id (q_id=q.q_id)  (cost=0.94 rows=3) (actual time=0.002..0.002 rows=1 loops=10947)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.09 sec)
```

3. Indexed using the following query. I chose to index on the creation date because the rest of the queries use foreign keys which are already indexed. We wanted to experiment if this could help with the performance. As you can see below it did not improve the runtime. As expected this did not increase the performance as creation date was not used in the query.

```
mysql> create index idx_c_creation_date on comments (c_creation_date);
```

```
| -> Limit: 15 row(s)  (actual time=89.651..89.655 rows=15 loops=1)
    -> Sort: total_responses DESC, limit input to 15 row(s) per chunk  (actual time=89.650..89.653 rows=15 loops=1)
        -> Stream results  (cost=24136.61 rows=34655) (actual time=13.337..87.205 rows=10000 loops=1)
            -> Group aggregate: count(distinct a.a_id), count(distinct c.c_id), count(distinct c.c_id), count(distinct a.a_id)  (cost=24136.6
1 rows=34655) (actual time=13.333..81.895 rows=10000 loops=1)
                -> Nested loop left join  (cost=20671.15 rows=34655) (actual time=13.310..69.169 rows=17683 loops=1)
                    -> Nested loop left join  (cost=4552.18 rows=13435) (actual time=13.293..38.695 rows=10947 loops=1)
                        -> Sort: q.q_id, q.title  (cost=1035.95 rows=8677) (actual time=13.262..14.628 rows=10000 loops=1)
                            -> Index scan on q using idx_q_title  (actual time=0.023..3.088 rows=10000 loops=1)
                        -> Covering index lookup on c using q_id (q_id=q.q_id)  (cost=0.25 rows=2) (actual time=0.002..0.002 rows=0 loops=100
00)
                    -> Covering index lookup on a using q_id (q_id=q.q_id)  (cost=0.94 rows=3) (actual time=0.002..0.003 rows=1 loops=10947)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.09 sec)
```

2. Running explain analyze on the **second** advanced query without indexing

```
-----------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=4654.31 rows=15) (actual time=46.401..46.489 rows=15 loops=1)
    -> Nested loop inner join  (cost=4654.31 rows=10052) (actual time=46.373..46.460 rows=15 loops=1)
        -> Sort: a.q_id, a.a_creation_date  (cost=1136.11 rows=10052) (actual time=12.222..12.231 rows=26 loops=1)
            -> Table scan on a  (cost=1136.11 rows=10052) (actual time=0.770..7.348 rows=10000 loops=1)
        -> Filter: <in_optimizer>((a.q_id,u.reputation),(a.q_id,u.reputation) in (select #2))  (cost=0.25 rows=1) (actual time=1.281..1.281 rows=1 loops=26)
            -> Single-row index lookup on u using PRIMARY (u_id=a.u_id)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=26)
            -> Select #2 (subquery in condition; run only once)
                -> Filter: ((a.q_id = `<materialized_subquery>`.q_id) and (u.reputation = `<materialized_subquery>`.max_reputation))  (cost=0.00..0.00 rows=0) (actual time=1.207..1
.207 rows=1 loops=27)
                    -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=1.206..1.206 rows=1 loops=27)
                        -> Index lookup on <materialized_subquery> using <auto_distinct_key> (q_id=a.q_id, max_reputation=u.reputation)  (actual time=1.206..1.206 rows=1 loops=27)
                            -> Materialize with deduplication  (cost=0.00..0.00 rows=0) (actual time=32.529..32.529 rows=3897 loops=1)
                                -> Table scan on <temporary>  (actual time=30.349..31.211 rows=3897 loops=1)
                                    -> Aggregate using temporary table  (actual time=30.347..30.347 rows=3897 loops=1)
                                        -> Nested loop inner join  (cost=4654.31 rows=10052) (actual time=0.053..21.636 rows=10000 loops=1)
                                            -> Table scan on a  (cost=1136.11 rows=10052) (actual time=0.041..7.324 rows=10000 loops=1)
                                            -> Single-row index lookup on u using PRIMARY (u_id=a.u_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=10000)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
1 row in set (0.05 sec)
```

1. Indexed using the following query. I chose to index on creation date because as seen earlier indexing on the foreign key did not help as it was already indexed and in the advanced query we are grouping by creation date. As you can see below it did improve the runtime from 0.05 seconds to 0.03 seconds.

```
mysql> CREATE INDEX idx_a_creation_date on answers (a_creation_date);
```

```
-----------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=4654.31 rows=15) (actual time=37.134..37.211 rows=15 loops=1)
    -> Nested loop inner join  (cost=4654.31 rows=10052) (actual time=37.133..37.208 rows=15 loops=1)
        -> Sort: a.q_id, a.a_creation_date  (cost=1136.11 rows=10052) (actual time=10.002..10.008 rows=26 loops=1)
            -> Table scan on a  (cost=1136.11 rows=10052) (actual time=0.037..5.644 rows=10000 loops=1)
        -> Filter: <in_optimizer>((a.q_id,u.reputation),(a.q_id,u.reputation) in (select #2))  (cost=0.25 rows=1) (actual time=1.046..1.046 rows=1 loops=26)
            -> Single-row index lookup on u using PRIMARY (u_id=a.u_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=26)
            -> Select #2 (subquery in condition; run only once)
                -> Filter: ((a.q_id = `<materialized_subquery>`.q_id) and (u.reputation = `<materialized_subquery>`.max_reputation))  (cost=0.00..0.00 rows=0) (actual time=0.992..0
.992 rows=1 loops=27)
                    -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=0.992..0.992 rows=1 loops=27)
                        -> Index lookup on <materialized_subquery> using <auto_distinct_key> (q_id=a.q_id, max_reputation=u.reputation)  (actual time=0.991..0.991 rows=1 loops=27)
                            -> Materialize with deduplication  (cost=0.00..0.00 rows=0) (actual time=26.735..26.735 rows=3897 loops=1)
                                -> Table scan on <temporary>  (actual time=24.800..25.496 rows=3897 loops=1)
                                    -> Aggregate using temporary table  (actual time=24.798..24.798 rows=3897 loops=1)
                                        -> Nested loop inner join  (cost=4654.31 rows=10052) (actual time=0.031..17.365 rows=10000 loops=1)
                                            -> Table scan on a  (cost=1136.11 rows=10052) (actual time=0.025..5.837 rows=10000 loops=1)
                                            -> Single-row index lookup on u using PRIMARY (u_id=a.u_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=10000)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
1 row in set (0.03 sec)
```

2. Indexing using the following query below. We chose to index on user reputation as reputation is being used in the queries to do various computations and it might increase in time cost. However, using this indexing did not increase the speed of the query. We feel this is because the query was already pretty fast and this indexing made little to no impact to it.

```
CREATE INDEX idx_reputation on users (reputation);
```

```
| -> Limit: 15 row(s)  (cost=4654.31 rows=15) (actual time=44.006..44.119 rows=15 loops=1)
    -> Nested loop inner join  (cost=4654.31 rows=10052) (actual time=44.004..44.115 rows=15 loops=1)
        -> Sort: a.q_id, a.a_creation_date  (cost=1136.11 rows=10052) (actual time=11.622..11.631 rows=26 loops=1)
            -> Table scan on a  (cost=1136.11 rows=10052) (actual time=0.056..6.697 rows=10000 loops=1)
        -> Filter: <in_optimizer>((a.q_id,u.reputation),(a.q_id,u.reputation) in (select #2))  (cost=0.25 rows=1) (actual time=1.249..1.249 rows=1 loops=26)
            -> Single-row index lookup on u using PRIMARY (u_id=a.u_id)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=26)
            -> Select #2 (subquery in condition; run only once)
                -> Filter: ((a.q_id = `<materialized_subquery>`.q_id) and (u.reputation = `<materialized_subquery>`.max_reputation))  (cost=0.00..0.00 rows=0) (actual time=1.160..1
.160 rows=1 loops=27)
                    -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=1.159..1.159 rows=1 loops=27)
                        -> Index lookup on <materialized_subquery> using <auto_distinct_key> (q_id=a.q_id, max_reputation=u.reputation)  (actual time=1.159..1.159 rows=1 loops=27)
                            -> Materialize with deduplication  (cost=0.00..0.00 rows=0) (actual time=31.240..31.240 rows=3897 loops=1)
                                -> Table scan on <temporary>  (actual time=28.775..29.726 rows=3897 loops=1)
                                    -> Aggregate using temporary table  (actual time=28.773..28.773 rows=3897 loops=1)
                                        -> Nested loop inner join  (cost=4654.31 rows=10052) (actual time=0.049..20.243 rows=10000 loops=1)
                                            -> Table scan on a  (cost=1136.11 rows=10052) (actual time=0.041..6.888 rows=10000 loops=1)
                                            -> Single-row index lookup on u using PRIMARY (u_id=a.u_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=10000)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------+
1 row in set (0.04 sec)
```

3. Indexing using the following query below. We chose to index on question id as the question id in the queries to do the group by and it might increase in time cost. This did not increase the query speed as q_id is also a foreign key and hence it is already indexed when running the query.

```
mysql> CREATE INDEX idx_q_id on answers (q_id);
```

```
| -> Limit: 15 row(s)  (cost=4654.31 rows=15) (actual time=38.506..38.581 rows=15 loops=1)
    -> Nested loop inner join  (cost=4654.31 rows=10052) (actual time=38.504..38.578 rows=15 loops=1)
        -> Sort: a.q_id, a.a_creation_date  (cost=1136.11 rows=10052) (actual time=10.438..10.444 rows=26 loops=1)
            -> Table scan on a  (cost=1136.11 rows=10052) (actual time=0.091..6.025 rows=10000 loops=1)
        -> Filter: <in_optimizer>((a.q_id,u.reputation),(a.q_id,u.reputation) in (select #2))  (cost=0.25 rows=1) (actual time=1.082..1.082 r
ows=1 loops=26)
            -> Single-row index lookup on u using PRIMARY (u_id=a.u_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=26)
            -> Select #2 (subquery in condition; run only once)
                -> Filter: ((a.q_id = `<materialized_subquery>`.q_id) and (u.reputation = `<materialized_subquery>`.max_reputation))  (cost=0
.00..0.00 rows=0) (actual time=1.026..1.026 rows=1 loops=27)
                    -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=1.025..1.025 rows=1 loops=27)
                        -> Index lookup on <materialized_subquery> using <auto_distinct_key> (q_id=a.q_id, max_reputation=u.reputation)  (act
ual time=1.025..1.025 rows=1 loops=27)
                            -> Materialize with deduplication  (cost=0.00..0.00 rows=0) (actual time=27.653..27.653 rows=3897 loops=1)
                                -> Table scan on <temporary>  (actual time=25.676..26.428 rows=3897 loops=1)
                                    -> Aggregate using temporary table  (actual time=25.674..25.674 rows=3897 loops=1)
                                        -> Nested loop inner join  (cost=4654.31 rows=10052) (actual time=0.032..18.054 rows=10000 loops=1)
                                            -> Table scan on a  (cost=1136.11 rows=10052) (actual time=0.026..6.046 rows=10000 loops=1)
                                            -> Single-row index lookup on u using PRIMARY (u_id=a.u_id)  (cost=0.25 rows=1) (actual time=0.00
1..0.001 rows=1 loops=10000)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------+
1 row in set (0.04 sec)
```

After this analysis we decided to not use indexing as it does not help with a significant increase in query performance. Because we have used foreign keys in our database implementation, a lot of the attributes are already indexed leading to quick speeds initially only. Another factor we thought of is that indexing might slow down our read, update and delete operations as the database will need to maintain the indexes whenever data changes and our project is very heavy on adding, deleting and updating posts.