Stage 3: Database Implementation and Indexing

Data Definition Language Commands

```
USE triviattack;

CREATE TABLE IF NOT EXISTS team (
        teamID INT NOT NULL,
        teamName VARCHAR(20),
        PRIMARY KEY(teamID)
);
CREATE TABLE IF NOT EXISTS user (
        userID INT NOT NULL,
        password VARCHAR(30) NOT NULL,
        Username VARCHAR(30) NOT NULL,
        profilePicURL VARCHAR(2083),
        PRIMARY KEY(userID)
);
CREATE TABLE IF NOT EXISTS teamMembership (
        teamID INT NOT NULL,
        userID INT NOT NULL,
        PRIMARY KEY(teamID, userID),
        FOREIGN KEY (teamID) REFERENCES team(teamID)
ON UPDATE CASCADE ON DELETE CASCADE,
   FOREIGN KEY (userID) REFERENCES user(userID)
ON UPDATE CASCADE ON DELETE CASCADE

);
CREATE TABLE IF NOT EXISTS quiz (
        quizID INT NOT NULL,
        PRIMARY KEY(quizID)
);

CREATE TABLE IF NOT EXISTS question(
        questionID INT NOT NULL,
        answer TEXT,
        questionText TEXT,
        subcategory VARCHAR(30),
        category VARCHAR(30),
        difficulty VARCHAR(25),
        PRIMARY KEY(questionID)
```

```
);

CREATE TABLE IF NOT EXISTS quizQuestions(
        quizID INT NOT NULL,
        questionID INT NOT NULL,
        PRIMARY KEY(quizID, questionID),
   FOREIGN KEY (quizID) REFERENCES quiz(quizID)
   ON UPDATE CASCADE ON DELETE CASCADE,
   FOREIGN KEY (questionID) REFERENCES question(questionID)
   ON UPDATE CASCADE ON DELETE CASCADE
);



CREATE TABLE IF NOT EXISTS response (
        responseID INT NOT NULL,
        isCorrect BIT,
        Date DATE,
        quizID INT NOT NULL,
        questionID INT NOT NULL,
        userID INT NOT NULL,
        PRIMARY KEY(userID, quizID, questionID),
   FOREIGN KEY (quizID) REFERENCES quiz(quizID)
   ON UPDATE CASCADE ON DELETE CASCADE,
   FOREIGN KEY (userID) REFERENCES user(userID)
   ON UPDATE CASCADE ON DELETE CASCADE,
   FOREIGN KEY (questionID) REFERENCES question(questionID)
   ON UPDATE CASCADE ON DELETE CASCADE
);
```

SQL Query Connection:

```
    connection = mysql.connector.connect(host='34.69.102.226',
                                         database='triviattack',
                                         user='root',
                                         password='X_hhZ2;u7J@;GFKv')
    if connection.is_connected():
        db_Info = connection.get_server_info()
        print("Connected to MySQL Server version ", db_Info)
        cursor = connection.cursor()
        cursor.execute("select database();")
        record = cursor.fetchone()
```

```
        print("You're connected to database: ", record)
```

Based on: https://pynative.com/python-mysql-database-connection/

Advanced SQL Queries

```
Advanced Query 1 (Gets the team(s) the user is on and the other members of
that team)

   select distinct u.Username, t.teamName
    from teamMembership m natural join user u natural join team t
    where teamID in (select teamID from teamMembership m1 where m1.userID
= 0001)
    group by t.teamName, u.Username
    order by t.teamName asc
```

```
+----------+----------+
| Username | teamName |
+----------+----------+
| user1829 | group691 |
| user2784 | group691 |
| user3304 | group691 |
| user3900 | group691 |
| user4734 | group691 |
| user6    | group691 |
| user218  | group94  |
| user2402 | group94  |
| user29   | group94  |
| user2921 | group94  |
| user3572 | group94  |
| user3899 | group94  |
| user4335 | group94  |
| user6    | group94  |
| user939  | group94  |
+----------+----------+
15 rows in set (0.00 sec)
```

Advanced Query 2 (Gets the average percent correct per members of a team)

```
    select mem.userID, p.percentCorrect, mem.teamID
    from teamMembership as mem natural join (select userID,
100*round(avg(isCorrect),2) as percentCorrect from response group by
userId) as p
    where mem.userID = p.userID and mem.teamID = 691
    order by percentCorrect desc
```

```
+--------+----------------+--------+
| userID | percentCorrect | teamID |
+--------+----------------+--------+
|    305 |          90.00 |    140 |
|    305 |          90.00 |    374 |
|   2358 |          90.00 |    677 |
|   2621 |          90.00 |    662 |
|   3448 |          90.00 |    445 |
|   4399 |          90.00 |    627 |
|    672 |          80.00 |    892 |
|   1587 |          80.00 |    553 |
|   2045 |          80.00 |    409 |
|   2208 |          80.00 |    402 |
|   2350 |          80.00 |    285 |
|   2994 |          80.00 |    880 |
|   3324 |          80.00 |    855 |
|   3977 |          80.00 |    315 |
|   3977 |          80.00 |    461 |
+--------+----------------+--------+
15 rows in set (0.00 sec)
```

Advanced Query 3 (Gets the average percent correct per category for a user)

```
    select p.percentCorrect, p.category
    from (select userID, 100*round(avg(isCorrect),2) as percentCorrect,
category from response natural join question group by userId, category) as
p
```

```
    where userID = 1829
    group by p.category
```

```
+----------------+----------------+
| percentCorrect | category       |
+----------------+----------------+
|          75.00 | Literature     |
|           0.00 | Social Science |
|         100.00 | Fine Arts      |
|          50.00 | History        |
|           0.00 | History        |
|          25.00 | Social Science |
|         100.00 | Literature     |
|           0.00 | Fine Arts      |
|           0.00 | Science        |
|           0.00 | Social Science |
|           0.00 | Fine Arts      |
|          50.00 | Science        |
|           0.00 | History        |
|         100.00 | Trash          |
|         100.00 | Social Science |
+----------------+----------------+
15 rows in set (0.02 sec)
```

Count Query:

```
select count(*) from team
```

Number of Rows for team

[(1000,)]

```
mysql> select count(*) from team;
+----------+
| count(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.01 sec)
```

```
select count(*) from user
```

Number of Rows for user

[(5000,)]

```
mysql> select count(*) from user;
+----------+
| count(*) |
+----------+
|     5000 |
+----------+
1 row in set (0.01 sec)
```

```
select count(*) from question
```

Number of Rows for question

[(2214,)]

```
mysql> select count(*) from question;
+----------+
| count(*) |
+----------+
|     2214 |
+----------+
1 row in set (0.00 sec)
```

```
select count(*) from response
```

Number of Rows for response

[(2800,)]

```
mysql> select count(*) from response;
+----------+
| count(*) |
+----------+
|     2800 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> select count(*) from teamMembership;
+----------+
| count(*) |
+----------+
|     5688 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> select count(*) from quizQuestions;
+----------+
| count(*) |
+----------+
|     5420 |
+----------+
1 row in set (0.00 sec)
```

Indexing Analysis

Advanced Query 1 with no index

```
Explain Analyze Query 1
('-> Sort: t.teamName, u.Username  (actual time=0.201..0.202 rows=15 loops=1)\n
 -> Table scan on <temporary>  (cost=0.23..2.64 rows=11) (actual time=0.001..0.002 rows=15 loops=1)\n
 -> Temporary table with deduplication  (cost=8.14..10.54 rows=11) (actual time=0.178..0.180 rows=15 loops=1)\n
 -> Nested loop inner join  (cost=6.77 rows=11) (actual time=0.046..0.095 rows=15 loops=1)\n
 -> Nested loop inner join  (cost=2.79 rows=11) (actual time=0.039..0.055 rows=15 loops=1)\n
 -> Nested loop inner join  (cost=1.15 rows=2) (actual time=0.031..0.037 rows=2 loops=1)\n
 -> Index lookup on m1 using userID (userID=6)  (cost=0.45 rows=2) (actual time=0.019..0.020 rows=2 loops=1)\n
 -> Single-row index lookup on t using PRIMARY (teamID=m1.teamID)  (cost=0.30 rows=1) (actual time=0.007..0.008 rows=1 loops=2)\n
 -> Index lookup on m using PRIMARY (teamID=m1.teamID)  (cost=0.54 rows=6) (actual time=0.006..0.008 rows=8 loops=2)\n
 -> Single-row index lookup on u using PRIMARY (userID=m.userID)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=15)\n',)
```

Advanced Query 1 with index on table (teamMembership) columns (userID)

```
('-> Sort: t.teamName, u.Username  (actual time=0.174..0.175 rows=15 loops=1)\n
-> Table scan on <temporary>  (cost=0.23..2.64 rows=11) (actual time=0.001..0.002 rows=15 loops=1)\n
-> Temporary table with deduplication  (cost=8.14..10.54 rows=11) (actual time=0.144..0.147 rows=15 loops=1)\n
-> Nested loop inner join  (cost=6.77 rows=11) (actual time=0.048..0.105 rows=15 loops=1)\n
-> Nested loop inner join  (cost=2.79 rows=11) (actual time=0.041..0.058 rows=15 loops=1)\n
-> Nested loop inner join  (cost=1.15 rows=2) (actual time=0.032..0.038 rows=2 loops=1)\n
-> Index lookup on m1 using userID (userID=6)  (cost=0.45 rows=2) (actual time=0.019..0.020 rows=2 loops=1)\n
-> Single-row index lookup on t using PRIMARY (teamID=m1.teamID)  (cost=0.30 rows=1) (actual time=0.008..0.008 rows=1 loops=2)\n
-> Index lookup on m using PRIMARY (teamID=m1.teamID)  (cost=0.54 rows=6) (actual time=0.006..0.009 rows=8 loops=2)\n
-> Single-row index lookup on u using PRIMARY (userID=m.userID)  (cost=0.26 rows=1) (actual time=0.003..0.003 rows=1 loops=15)\n',)
```

Advanced Query 1 with index on table (user) columns (Username)

```
('-> Sort: t.teamName, u.Username  (actual time=0.282..0.283 rows=15 loops=1)\n
-> Table scan on <temporary>  (cost=0.23..2.64 rows=11) (actual time=0.001..0.003 rows=15 loops=1)\n
-> Temporary table with deduplication  (cost=8.14..10.54 rows=11) (actual time=0.256..0.259 rows=15 loops=1)\n
-> Nested loop inner join  (cost=6.77 rows=11) (actual time=0.042..0.213 rows=15 loops=1)\n
-> Nested loop inner join  (cost=2.79 rows=11) (actual time=0.036..0.175 rows=15 loops=1)\n
-> Nested loop inner join  (cost=1.15 rows=2) (actual time=0.028..0.034 rows=2 loops=1)\n
-> Index lookup on m1 using userID (userID=6)  (cost=0.45 rows=2) (actual time=0.019..0.020 rows=2 loops=1)\n
-> Single-row index lookup on t using PRIMARY (teamID=m1.teamID)  (cost=0.30 rows=1) (actual time=0.006..0.006 rows=1 loops=2)\n
-> Index lookup on m using PRIMARY (teamID=m1.teamID)  (cost=0.54 rows=6) (actual time=0.067..0.070 rows=8 loops=2)\n
-> Single-row index lookup on u using PRIMARY (userID=m.userID)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=15)\n',)
```

Advanced Query 1 with index on table (team) columns (teamName)

```
('-> Sort: t.teamName, u.Username  (actual time=0.138..0.139 rows=15 loops=1)\n
-> Table scan on <temporary>  (cost=0.23..2.64 rows=11) (actual time=0.001..0.003 rows=15 loops=1)\n
-> Temporary table with deduplication  (cost=8.14..10.54 rows=11) (actual time=0.115..0.118 rows=15 loops=1)\n
-> Nested loop inner join  (cost=6.77 rows=11) (actual time=0.038..0.081 rows=15 loops=1)\n
-> Nested loop inner join  (cost=2.79 rows=11) (actual time=0.032..0.047 rows=15 loops=1)\n
-> Nested loop inner join  (cost=1.15 rows=2) (actual time=0.026..0.031 rows=2 loops=1)\n
-> Index lookup on m1 using userID (userID=6)  (cost=0.45 rows=2) (actual time=0.017..0.018 rows=2 loops=1)\n
-> Single-row index lookup on t using PRIMARY (teamID=m1.teamID)  (cost=0.30 rows=1) (actual time=0.006..0.006 rows=1 loops=2)\n
-> Index lookup on m using PRIMARY (teamID=m1.teamID)  (cost=0.54 rows=6) (actual time=0.005..0.007 rows=8 loops=2)\n
-> Single-row index lookup on u using PRIMARY (userID=m.userID)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=15)\n',)
```

Analysis:
We needed to create an index on userID or teamID or both on the teamMembership table or Username on the user table or teamName on the team table, because those were the columns that were integral to the search in Advanced Query 1. After experimenting with indices using each attribute individually, we found that using the default primary key index resulted in the best performance. Note the marginal improvement on the Single-row index lookup on t and the index lookup on m1 and m is on the order of 10^-3 seconds over the other indexing methods. We felt this did not warrant creating an index for our final table design. We conclude the index did not help because the database size is relatively small and the nature of our query is looking up single indices in the table, so creating an index based on the primary keys would have minimal impact. We did also note that the sort was improved on the teamName index, but the improvement was marginal compared to the time spent on other operations.

Advanced Query 3 with no index

```
('-> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=0.000..0.001 rows=6 loops=1)\n
-> Temporary table with deduplication  (cost=5.00..5.00 rows=0) (actual time=0.127..0.128 rows=6 loops=1)\n
-> Table scan on p  (cost=2.50..2.50 rows=0) (actual time=0.000..0.001 rows=6 loops=1)\n
-> Materialize  (cost=2.50..2.50 rows=0) (actual time=0.107..0.108 rows=6 loops=1)\n
-> Table scan on <temporary>  (actual time=0.001..0.002 rows=6 loops=1)\n
-> Aggregate using temporary table  (actual time=0.091..0.092 rows=6 loops=1)\n
-> Nested loop inner join  (cost=4.75 rows=10) (actual time=0.032..0.061 rows=10 loops=1)\n
-> Index lookup on response using PRIMARY (userID=1829)  (cost=1.25 rows=10) (actual time=0.021..0.025 rows=10 loops=1)\n
-> Single-row index lookup on question using PRIMARY (questionID=response.questionID)  (cost=0.26 rows=1) (actual time=0.003..0.003 rows=1 loops=10)
```

Advanced Query 3 with index on table (response) columns (userID)

```
('-> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=0.000..0.001 rows=6 loops=1)\n
-> Temporary table with deduplication  (cost=5.00..5.00 rows=0) (actual time=0.120..0.121 rows=6 loops=1)\n
-> Table scan on p  (cost=2.50..2.50 rows=0) (actual time=0.000..0.001 rows=6 loops=1)\n
-> Materialize  (cost=2.50..2.50 rows=0) (actual time=0.096..0.097 rows=6 loops=1)\n
-> Table scan on <temporary>  (actual time=0.001..0.001 rows=6 loops=1)\n
-> Aggregate using temporary table  (actual time=0.082..0.083 rows=6 loops=1)\n
-> Nested loop inner join  (cost=4.75 rows=10) (actual time=0.028..0.052 rows=10 loops=1)\n
-> Index lookup on response using PRIMARY (userID=1829)  (cost=1.25 rows=10) (actual time=0.019..0.022 rows=10 loops=1)\n
-> Single-row index lookup on question using PRIMARY (questionID=response.questionID)  (cost=0.26 rows=1) (actual time=0.003..0.003 rows=1 loops=10)
```

Advanced Query 3 with index on table (question) column (category)

```
('-> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=0.000..0.001 rows=6 loops=1)\n
-> Temporary table with deduplication  (cost=5.00..5.00 rows=0) (actual time=0.135..0.136 rows=6 loops=1)\n
-> Table scan on p  (cost=2.50..2.50 rows=0) (actual time=0.000..0.001 rows=6 loops=1)\n
-> Materialize  (cost=2.50..2.50 rows=0) (actual time=0.107..0.108 rows=6 loops=1)\n
-> Table scan on <temporary>  (actual time=0.001..0.001 rows=6 loops=1)\n
-> Aggregate using temporary table  (actual time=0.094..0.095 rows=6 loops=1)\n
-> Nested loop inner join  (cost=4.75 rows=10) (actual time=0.026..0.065 rows=10 loops=1)\n
-> Index lookup on response using PRIMARY (userID=1829)  (cost=1.25 rows=10) (actual time=0.019..0.022 rows=10 loops=1)\n
```

Advanced Query 3 with index on table (response) column (isCorrect)

```
('-> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=0.000..0.001 rows=6 loops=1)\n
-> Temporary table with deduplication  (cost=5.00..5.00 rows=0) (actual time=0.156..0.157 rows=6 loops=1)\n
-> Table scan on p  (cost=2.50..2.50 rows=0) (actual time=0.000..0.001 rows=6 loops=1)\n
-> Materialize  (cost=2.50..2.50 rows=0) (actual time=0.133..0.134 rows=6 loops=1)\n
-> Table scan on <temporary>  (actual time=0.001..0.002 rows=6 loops=1)\n
-> Aggregate using temporary table  (actual time=0.117..0.118 rows=6 loops=1)\n
-> Nested loop inner join  (cost=4.75 rows=10) (actual time=0.030..0.087 rows=10 loops=1)\n
-> Index lookup on response using PRIMARY (userID=1829)  (cost=1.25 rows=10) (actual time=0.020..0.024 rows=10 loops=1)\n
-> Single-row index lookup on question using PRIMARY (questionID=response.questionID)  (cost=0.26 rows=1) (actual time=0.006..0.006 rows=1 loops=10)
```

Analysis:

We needed to create an index on userID or category or isCorrect because those were the columns that were integral to the search in Advanced Query 3. After experimenting with indices using each attribute individually, we found that indexing on isCorrect/category resulted in a decrease in performance, while userID resulted in marginal performance increases. However, we noticed that the improvement from userID was not significant over the indexless query (note the marginal improvement on the aggregate and the index lookup on response using primary is on the order of 10^-2 seconds.) We felt this did not warrant creating an index for our final table design. We conclude the index did not help because the database size is relatively small and the nature of our query is looking up single indices in the table, so creating an index based on the primary keys would have minimal impact.