Stage 3: Database Design

Part1.1)

Part 1.2)

```sql
CREATE TABLE User (
      UserID int,
      Username varchar(255) NOT NULL,
      Email varchar(255),
      PasswordHash varchar(255) NOT NULL,
      RegistrationDate DATE,
      PRIMARY KEY (UserID)
);
```

```sql
CREATE TABLE UserSubmission (
      SubmissionID int,
      UserID int,
      QuestionText varchar(255) NOT NULL,
      CorrectAnswer varchar(255) NOT NULL,
      IncorrectAns1 varchar(255) NOT NULL,
```

```
      IncorrectAns2 varchar(255) NOT NULL,
      IncorrectAns3 varchar(255) NOT NULL,
      Status TINYINT(1),
      SubmissionDate DATE,
      CategoryID int,
      PRIMARY KEY (SubmissionID),
      FOREIGN KEY(CategoryID) REFERENCES Category(CategoryID),
      FOREIGN KEY(UserID) REFERENCES User(UserID)
);
```

```
CREATE TABLE Question (
      QuestionID int,
      QuestionText varchar(255) NOT NULL,
      CorrectAnswer varchar(255) NOT NULL,
      IncorrectAns1 varchar(255) NOT NULL,
      IncorrectAns2 varchar(255) NOT NULL,
      IncorrectAns3 varchar(255) NOT NULL,
      Difficulty int,
      CategoryID int,
      PRIMARY KEY (QuestionID),
      FOREIGN KEY(CategoryID) REFERENCES Category(CategoryID)
);
```

```
CREATE TABLE Category (
      CategoryID int,
      Type varchar(3),
      Subcategory varchar(255),
      PRIMARY KEY (CategoryID)
);
```

```
CREATE TABLE UserHighScore (
      UserID int,
      CategoryID int,
      TriviaMode VARCHAR(10) NOT NULL,
      HighScore int,
      PRIMARY KEY (UserID, CategoryID, TriviaMode),
      FOREIGN KEY (UserID) REFERENCES User(UserID),
      FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID)
);
```

Part 1.3)

```
mysql> select count(*) from User;
+----------+
| count(*) |
+----------+
|     1001 |
+----------+
1 row in set (0.02 sec)

mysql>
```

```
mysql> select count(*) from Question;
+----------+
| count(*) |
+----------+
|     1001 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> select count(*) from UserHighScore;
+----------+
| count(*) |
+----------+
|     1001 |
+----------+
1 row in set (0.00 sec)
```

Part 1.4)

Query #1:

```sql
-- retrieves the high score and username for the first 15 users on the NFL category whose score
is greater than the average score of category "NFL"
Select u1.Username, uhs1.HighScore
From UserHighScore uhs1
Join User u1 On u1.UserID = uhs1.UserID
Where uhs1.CategoryID In (
 Select CategoryID From Category Where Type = 'NFL'
)
```

```
 And uhs1.HighScore >= (
   Select Avg(uhs2.HighScore)
   From UserHighScore uhs2
   Where uhs2.CategoryID In (
     Select CategoryID From Category Where Type = 'NFL'
   )
 )
Order By uhs1.HighScore Desc
Limit 15;
```

```
mysql> Select u1.Username, uhs1.HighScore
    -> From UserHighScore uhs1
    -> Join User u1 On u1.UserID = uhs1.UserID
    -> Where uhs1.CategoryID In (
    ->   Select CategoryID From Category Where Type = 'NFL'
    -> )
    ->   And uhs1.HighScore >= (
    ->     Select Avg(uhs2.HighScore)
    ->     From UserHighScore uhs2
    ->     Where uhs2.CategoryID In (
    ->       Select CategoryID From Category Where Type = 'NFL'
    ->     )
    ->   )
    -> Order By uhs1.HighScore Desc
    -> Limit 15;
+--------------+-----------+
| Username     | HighScore |
+--------------+-----------+
| Zavier123    |        10 |
| Joey123      |        10 |
| Zechariah123 |        10 |
| Harold123    |        10 |
| Shmuel123    |        10 |
| Osiris123    |        10 |
| Henrik123    |        10 |
| Eliezer123   |        10 |
| Osman123     |        10 |
| Cullen123    |        10 |
| Hollis123    |        10 |
| Veer123      |        10 |
| Adler123     |        10 |
| Jasiel123    |        10 |
| Imran123     |        10 |
+--------------+-----------+
15 rows in set (0.00 sec)
```

Query #2:

```
-- query unifies official questions from the Question table where the question difficulty is 1
with approved user submitted questions from UserSubmission on the NFL category
Select *
```

```sql
From (
  Select
    'Official' As QuestionSource,
    q.QuestionID As ID,
    q.QuestionText,
    q.CorrectAnswer,
    q.IncorrectAns1,
    q.IncorrectAns2,
    q.IncorrectAns3
  From Question q
  Join Category c On q.CategoryID = c.CategoryID
  Where q.Difficulty = 1

  Union All

  Select
    'UserSubmitted' As QuestionSource,
    us.SubmissionID As ID,
    us.QuestionText,
    us.CorrectAnswer,
    us.IncorrectAns1,
    us.IncorrectAns2,
    us.IncorrectAns3
  From UserSubmission us
  Join Category c On us.CategoryID = c.CategoryID
  Where c.Type = 'NFL' And us.Status = 1
) As CombinedQuestions
Limit 15;
```

```
mysql> Select *
    -> From (
    ->   Select
    ->     'Official' As QuestionSource,
    ->     q.QuestionID As ID,
    ->     q.QuestionText,
    ->     q.CorrectAnswer,
    ->     q.IncorrectAns1,
    ->     q.IncorrectAns2,
    ->     q.IncorrectAns3
    ->   From Question q
    ->   Join Category c On q.CategoryID = c.CategoryID
    ->   Where q.Difficulty = 1
    ->
    ->   Union All
    ->
    ->   Select
    ->     'UserSubmitted' As QuestionSource,
    ->     us.SubmissionID As ID,
    ->     us.QuestionText,
    ->     us.CorrectAnswer,
    ->     us.IncorrectAns1,
    ->     us.IncorrectAns2,
    ->     us.IncorrectAns3
    ->   From UserSubmission us
    ->   Join Category c On us.CategoryID = c.CategoryID
    ->   Where c.Type = 'NFL' And us.Status = 1
    -> ) As CombinedQuestions
    -> Limit 15;
```

QuestionSource	ID	QuestionText	CorrectAnswer	IncorrectAns1	IncorrectAns2	IncorrectAns3
Official	3	Who holds the record for most passing yards in NFL history?	Tom Brady	Peyton Manning	Drew Brees	Brett Favre
Official	9	Who was the first overall pick in the 2020 NFL Draft?	Joe Burrow	Chase Young	Tua Tagovailoa	Justin Herbert
Official	13	What is the name of the trophy awarded to the college football national champion?	College Football Playoff National Championship Trophy	Heisman Trophy	Lombardi Trophy	Sugar Bowl Trophy
Official	15	Who was the first overall pick in the 2020 NFL Draft?	Joe Burrow	Chase Young	Tua Tagovailoa	Justin Herbert
Official	17	Who won the first Super Bowl?	Green Bay Packers	Dallas Cowboys	New York Giants	Pittsburgh Steelers
Official	21	Which NFL team is known as 'America's Team'?	Dallas Cowboys	New England Patriots	Green Bay Packers	Pittsburgh Steelers
Official	24	Who won the Heisman Trophy in 2019?	Joe Burrow	Kyler Murray	Tua Tagovailoa	Jalen Hurts
Official	25	Who was the first overall pick in the 2020 NFL Draft?	Joe Burrow	Chase Young	Tua Tagovailoa	Justin Herbert
Official	27	Which team won the 2021 College Football Playoff National Championship?	Georgia Bulldogs	Alabama Crimson Tide	Clemson Tigers	Ohio State Buckeyes
Official	30	Which player has the most career rushing yards in NFL history?	Emmitt Smith	Walter Payton	Barry Sanders	Adrian Peterson
Official	34	Who holds the record for most passing yards in NFL history?	Tom Brady	Peyton Manning	Drew Brees	Brett Favre
Official	35	Who won the first Super Bowl?	Green Bay Packers	Dallas Cowboys	New York Giants	Pittsburgh Steelers
Official	38	What is the name of the trophy awarded to the college football national champion?	College Football Playoff National Championship Trophy	Heisman Trophy	Lombardi Trophy	Sugar Bowl Trophy
Official	41	Who holds the record for most passing yards in NFL history?	Tom Brady	Peyton Manning	Drew Brees	Brett Favre
Official	42	Who won the Heisman Trophy in 2019?	Joe Burrow	Kyler Murray	Tua Tagovailoa	Jalen Hurts

15 rows in set (0.01 sec)

Query #3:

```sql
-- retrieves the username and high score for the top 15 users on standard mode for the NFL
```

```sql
category whose score are greater than average score on standard mode for the NFL
Select u1.Username, uhs1.HighScore
From UserHighScore uhs1
Join User u1 On uhs1.UserID = u1.UserID
Where uhs1.TriviaMode = 'Standard'
 And uhs1.CategoryID In (
   Select CategoryID From Category Where Type = 'NFL'
 )
 And uhs1.HighScore >= (
   Select Avg(uhs2.HighScore)
   From UserHighScore uhs2
   Where uhs2.TriviaMode = 'Standard'
     And uhs2.CategoryID In (
       Select CategoryID From Category Where Type = 'NFL'
     )
 )
Order By uhs1.HighScore Desc
Limit 15;
```

```
mysql> Select u1.Username, uhs1.HighScore
    -> From UserHighScore uhs1
    -> Join User u1 On uhs1.UserID = u1.UserID
    -> Where uhs1.TriviaMode = 'Standard'
    ->   And uhs1.CategoryID In (
    ->     Select CategoryID From Category Where Type = 'NFL'
    ->   )
    ->   And uhs1.HighScore >= (
    ->     Select Avg(uhs2.HighScore)
    ->     From UserHighScore uhs2
    ->     Where uhs2.TriviaMode = 'Standard'
    ->       And uhs2.CategoryID In (
    ->         Select CategoryID From Category Where Type = 'NFL'
    ->       )
    ->   )
    -> Order By uhs1.HighScore Desc
    -> Limit 15;
+--------------+-----------+
| Username     | HighScore |
+--------------+-----------+
| Zavier123    |        10 |
| Joey123      |        10 |
| Zechariah123 |        10 |
| Harold123    |        10 |
| Shmuel123    |        10 |
| Osiris123    |        10 |
| Henrik123    |        10 |
| Eliezer123   |        10 |
| Osman123     |        10 |
| Cullen123    |        10 |
| Hollis123    |        10 |
| Veer123      |        10 |
| Adler123     |        10 |
| Jasiel123    |        10 |
| Imran123     |        10 |
+--------------+-----------+
15 rows in set (0.00 sec)
```

Query #4:

```sql
-- retrieves the categories, trivia modes, and high scores for the first 15 users who have a
high score greater than the average high score in their trivia mode and registered after
2005-01-01
select c1.Type as Category, c1.Subcategory, uhs1.TriviaMode, uhs1.HighScore
from UserHighScore uhs1
join Category c1 on uhs1.CategoryID = c1.CategoryID
join User u on uhs1.UserID = u.UserID
where uhs1.HighScore >= (
  select avg(uhs2.HighScore)
  from UserHighScore uhs2
  where uhs2.TriviaMode = uhs1.TriviaMode
```

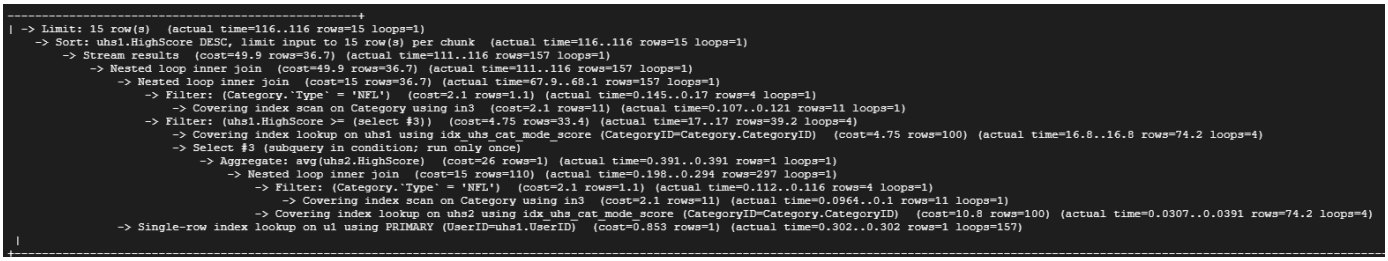
```
)
and u.RegistrationDate > '2005-01-01'
order by c1.Type, uhs1.TriviaMode
limit 15;
```

```
mysql> select c1.Type as Category, c1.Subcategory, uhs1.TriviaMode, uhs1.HighScore
    -> from UserHighScore uhs1
    -> join Category c1 on uhs1.CategoryID = c1.CategoryID
    -> join User u on uhs1.UserID = u.UserID
    -> where uhs1.HighScore >= (
    ->   select avg(uhs2.HighScore)
    ->   from UserHighScore uhs2
    ->   where uhs2.TriviaMode = uhs1.TriviaMode
    -> )
    -> and u.RegistrationDate > '2005-01-01'
    -> order by c1.Type, uhs1.TriviaMode
    -> limit 15;
+----------+-------------+------------+------------+
| Category | Subcategory | TriviaMode | HighScore  |
+----------+-------------+------------+------------+
| CFB      | 663         | Standard   |          7 |
| CFB      | 316         | Standard   |          7 |
| CFB      | 874         | Standard   |          7 |
| CFB      | 892         | Standard   |          7 |
| CFB      | 892         | Standard   |          7 |
| CFB      | 663         | Standard   |          7 |
| CFB      | 298         | Standard   |          7 |
| CFB      | 298         | Standard   |          7 |
| CFB      | 955         | Standard   |          7 |
| CFB      | 298         | Standard   |          7 |
| CFB      | 874         | Standard   |          7 |
| CFB      | 874         | Standard   |          7 |
| CFB      | 316         | Standard   |          7 |
| CFB      | 892         | Standard   |          7 |
| CFB      | 298         | Standard   |          7 |
+----------+-------------+------------+------------+
15 rows in set (0.05 sec)
```

Part 2.1)

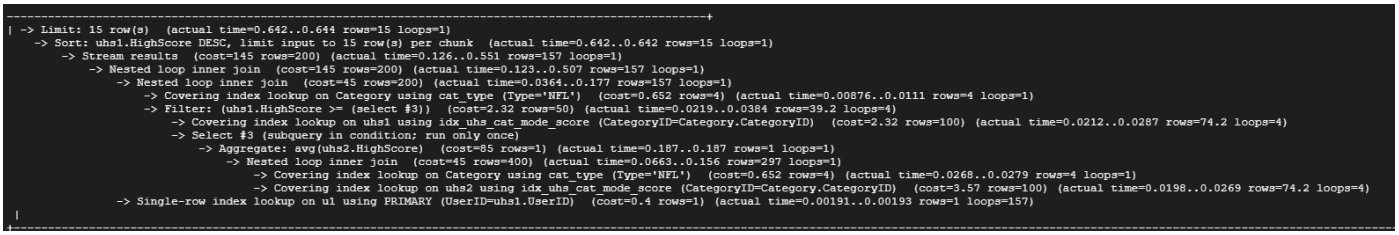
Index Analysis Query #1:

Default Index Performance:



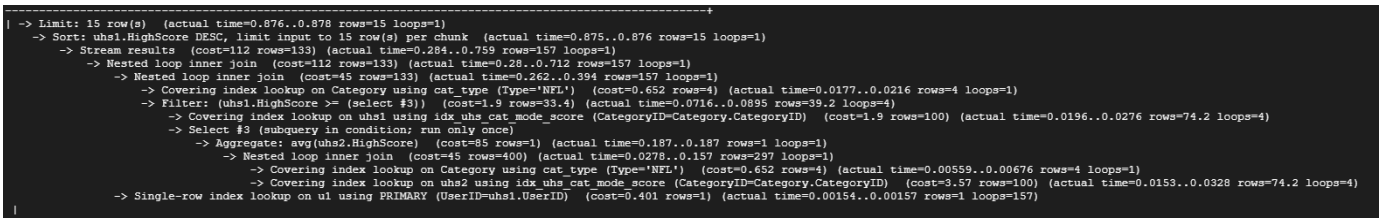
Total Cost = 49.9

Index on UserHighScore(HighScore Desc) Performance:



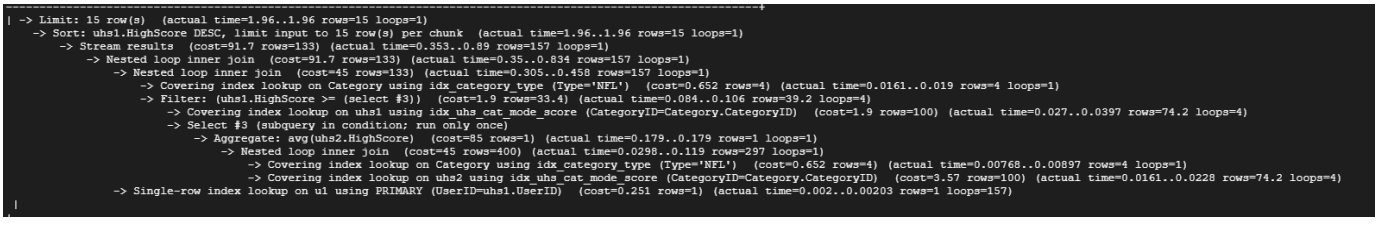
Total Cost = 145

Index on Category(Type) Performance:



Total Cost = 112

Index on User(UserID, Useranme) Performance:



Total Cost = 91.7

Indexing evaluation:

Based on our designed advanced query, we tried to test and index our query on UserHighScore(HighScore Desc), Category(Type), and User(UserID, Useranme) based on our joined attributes and attributes in the where clause. Compared to the result, we found that none of our indexes can reduce performance costs. Our intuitive thinking is that right now we don't have such realistic user data, and our HighScore is only scaled from 0 to 10, which actually might worsen the performance from indexing HighScore. For the similar reason, the category type has very low cardinality to index. Finally, we tried indexing using a composite key (UserID, Useranme) to index,

and the result was much better than the previous two. However, we think the reason it is still worse than the default indexing is because the UserID and Username are auto-generated and not varied enough for indexing.

Index Analysis Query #2:

Default Index Performance:

```
| -> Append  (cost=223 rows=110) (actual time=0.687..4.42 rows=556 loops=1)
    -> Stream results  (cost=201 rows=100) (actual time=0.684..2.73 rows=349 loops=1)
        -> Nested loop inner join  (cost=201 rows=100) (actual time=0.669..2.44 rows=349 loops=1)
            -> Covering index scan on c using in3  (cost=2.1 rows=11) (actual time=0.0796..0.0848 rows=11 loops=1)
            -> Filter: (q.Difficulty = 1)  (cost=9.08 rows=9.1) (actual time=0.0596..0.187 rows=31.7 loops=11)
                -> Index lookup on q using idx_status_categoryid (CategoryID=c.CategoryID)  (cost=9.08 rows=91) (actual time=0.0584..0.179 rows=91 loops=11)
    -> Stream results  (cost=22 rows=10) (actual time=0.711..1.64 rows=207 loops=1)
        -> Nested loop inner join  (cost=22 rows=10) (actual time=0.702..1.5 rows=207 loops=1)
            -> Filter: (c.`Type` = 'NFL')  (cost=2.1 rows=1.1) (actual time=0.0816..0.0998 rows=4 loops=1)
                -> Covering index scan on c using in3  (cost=2.1 rows=11) (actual time=0.0736..0.0892 rows=11 loops=1)
            -> Filter: (us.`Status` = 1)  (cost=9.83 rows=9.1) (actual time=0.191..0.345 rows=51.8 loops=4)
                -> Index lookup on us using CategoryID (CategoryID=c.CategoryID)  (cost=9.83 rows=91) (actual time=0.189..0.335 rows=101 loops=4)
```

Total Cost = 223

Index on Category(Type) Performance:

```
| -> Append  (cost=211 rows=136) (actual time=0.929..3.47 rows=556 loops=1)
    -> Stream results  (cost=138 rows=100) (actual time=0.928..2.19 rows=349 loops=1)
        -> Nested loop inner join  (cost=138 rows=100) (actual time=0.92..1.93 rows=349 loops=1)
            -> Filter: ((q.Difficulty = 1) and (q.CategoryID is not null))  (cost=103 rows=100) (actual time=0.717..1.43 rows=349 loops=1)
                -> Table scan on q  (cost=103 rows=1001) (actual time=0.698..1.32 rows=1001 loops=1)
            -> Single-row covering index lookup on c using PRIMARY (CategoryID=q.CategoryID)  (cost=0.251 rows=1) (actual time=0.00116..0.00119 rows=1 loops=349)
    -> Stream results  (cost=73.1 rows=36.4) (actual time=0.4..1.24 rows=207 loops=1)
        -> Nested loop inner join  (cost=73.1 rows=36.4) (actual time=0.394..1.1 rows=207 loops=1)
            -> Covering index lookup on c using cat_type (Type='NFL')  (cost=0.652 rows=4) (actual time=0.0442..0.0475 rows=4 loops=1)
            -> Filter: (us.`Status` = 1)  (cost=9.23 rows=9.1) (actual time=0.116..0.259 rows=51.8 loops=4)
                -> Index lookup on us using CategoryID (CategoryID=c.CategoryID)  (cost=9.23 rows=91) (actual time=0.114..0.251 rows=101 loops=4)
```

Total Cost = 211

Index on UserSubmission(Status) Performance:

```
| -> Append  (cost=159 rows=150) (actual time=0.139..3.72 rows=556 loops=1)
    -> Stream results  (cost=138 rows=100) (actual time=0.138..1.99 rows=349 loops=1)
        -> Nested loop inner join  (cost=138 rows=100) (actual time=0.13..1.6 rows=349 loops=1)
            -> Filter: ((q.Difficulty = 1) and (q.CategoryID is not null))  (cost=103 rows=100) (actual time=0.0844..1.12 rows=349 loops=1)
                -> Table scan on q  (cost=103 rows=1001) (actual time=0.0769..0.981 rows=1001 loops=1)
            -> Single-row covering index lookup on c using PRIMARY (CategoryID=q.CategoryID)  (cost=0.251 rows=1) (actual time=0.00102..0.00105 rows=1 loops=349)
    -> Stream results  (cost=21.3 rows=49.8) (actual time=0.298..1.68 rows=207 loops=1)
        -> Nested loop inner join  (cost=21.3 rows=49.8) (actual time=0.289..1.43 rows=207 loops=1)
            -> Filter: (c.`Type` = 'NFL')  (cost=1.35 rows=1.1) (actual time=0.149..0.161 rows=4 loops=1)
                -> Covering index scan on c using in3  (cost=1.35 rows=11) (actual time=0.0374..0.046 rows=11 loops=1)
            -> Filter: (us.`Status` = 1)  (cost=13.1 rows=45.3) (actual time=0.0816..0.311 rows=51.8 loops=4)
                -> Index lookup on us using CategoryID (CategoryID=c.CategoryID)  (cost=13.1 rows=91) (actual time=0.0801..0.298 rows=101 loops=4)
```

Total Cost = 159

Index on Question(Difficulty) Performance:

```
| -> Append  (cost=187 rows=359) (actual time=0.167..4.54 rows=556 loops=1)
    -> Stream results  (cost=166 rows=349) (actual time=0.166..1.96 rows=349 loops=1)
        -> Nested loop inner join  (cost=166 rows=349) (actual time=0.16..1.55 rows=349 loops=1)
            -> Filter: (q.CategoryID is not null)  (cost=43.9 rows=349) (actual time=0.146..1.01 rows=349 loops=1)
                -> Index lookup on q using q_difficulty (Difficulty=1)  (cost=43.9 rows=349) (actual time=0.144..0.975 rows=349 loops=1)
            -> Single-row covering index lookup on c using PRIMARY (CategoryID=q.CategoryID)  (cost=0.25 rows=1) (actual time=0.00122..0.00127 rows=1 loops=349)
    -> Stream results  (cost=21.3 rows=10) (actual time=0.268..2.52 rows=207 loops=1)
        -> Nested loop inner join  (cost=21.3 rows=10) (actual time=0.262..2.27 rows=207 loops=1)
            -> Filter: (c.`Type` = 'NFL')  (cost=1.35 rows=1.1) (actual time=0.0239..0.0346 rows=4 loops=1)
                -> Covering index scan on c using in3  (cost=1.35 rows=11) (actual time=0.0165..0.0238 rows=11 loops=1)
            -> Filter: (us.`Status` = 1)  (cost=9.83 rows=9.1) (actual time=0.11..0.553 rows=51.8 loops=4)
                -> Index lookup on us using CategoryID (CategoryID=c.CategoryID)  (cost=9.83 rows=91) (actual time=0.108..0.33 rows=101 loops=4)
```

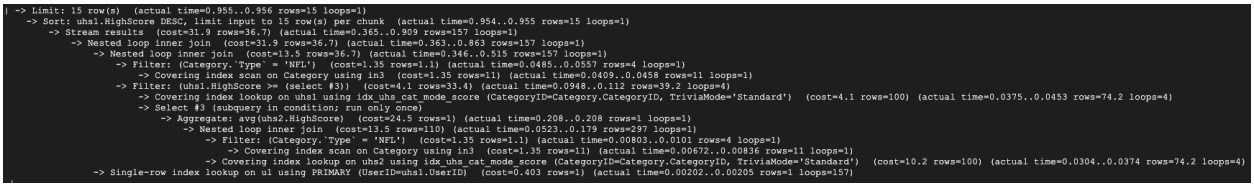
Total Cost = 187

Indexing Evaluation:
All three indexes lower the total cost as they are all helpful for faster querying. Indexing on the Status attribute in the UserSubmission table is the most effective. Our guess is that since Status is of type TINYINT(1) (boolean), making it an index helps eliminate all submitted questions with status of 0 quickly, making the query the fastest. Although Category(Type) also has only two options, it is

of type VARCHAR, which is slower to index and query. And Question(Difficulty) ranges from 0 to 3, which is less effective for indexing. Thus, we choose to have the final design of having an index on UserSubmission(Status).

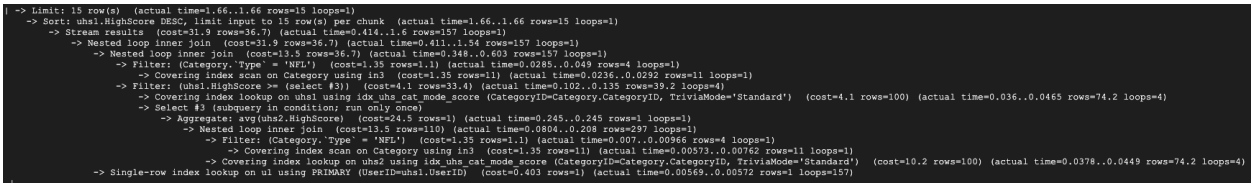
Index Analysis Query #3:

Default Index Performance:



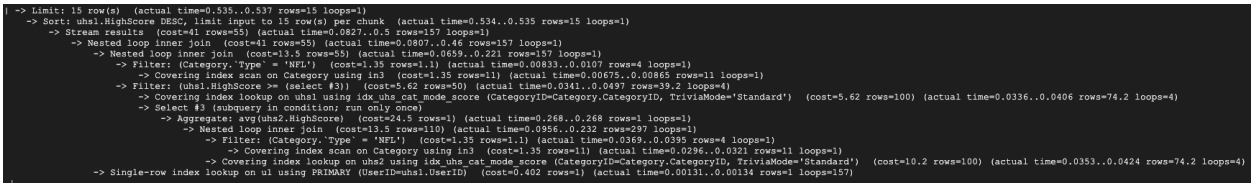
Total Cost = 31.9

Index on UserHighScore(TriviaMode) Performance:



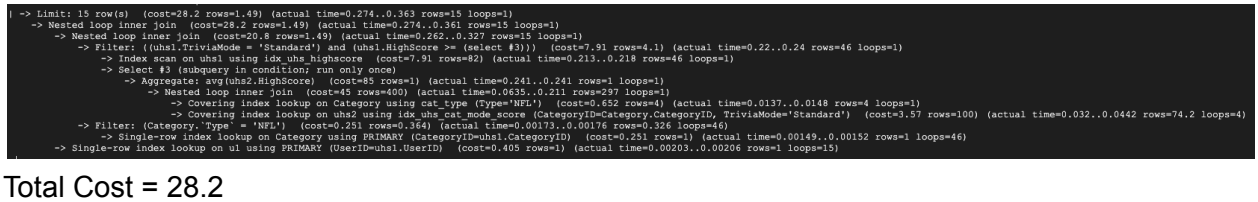
Total Cost = 31.9

Index on UserHighScore(HighScore) Performance:



Total Cost = 41

Index on Category(Type) Performance:



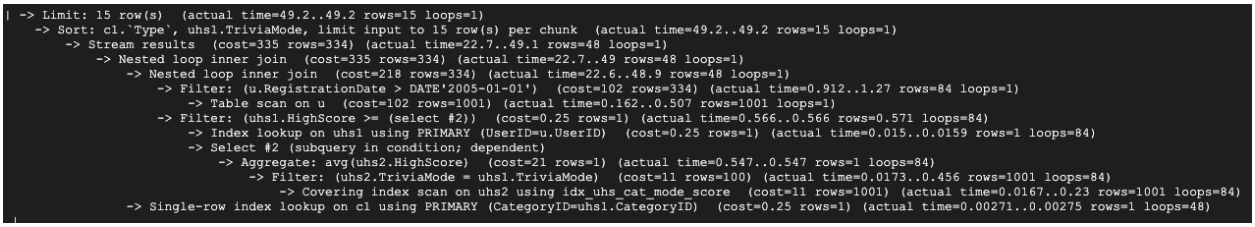
Total Cost = 28.2

Indexing Evaluation:
Indexing on the Type attribute in the Category table is most effective. In this case, indexing on UserHighScore(TriviaMode) had no effect on the total cost. Indexing on TriviaMode follows relatively the same computation as the default index. Indexing on UserHighScore(HighScore) is slightly different though, but this increased the time. Indexing with this increases the cost a bit since the high scores are only between 1 and 10 and this is being done within a subquery. Category(Type)

has the best performance even though the Type variable only has 3 options. This is the best since it is using the Category table to index values which means scanning UserHighScore is easier as there is the index in the Category table there.

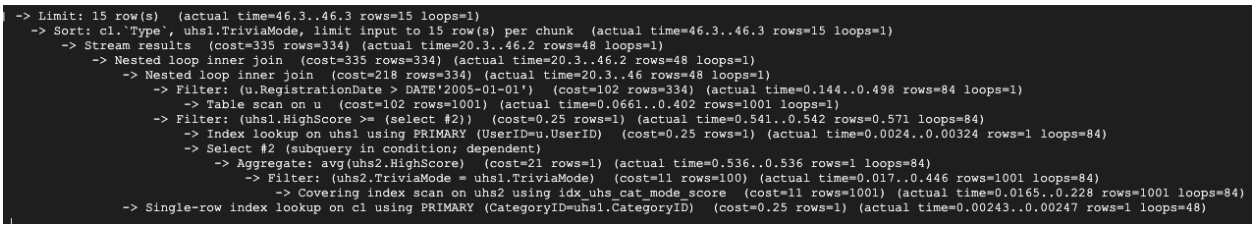
Index Analysis Query #4:

Default Index Performance:



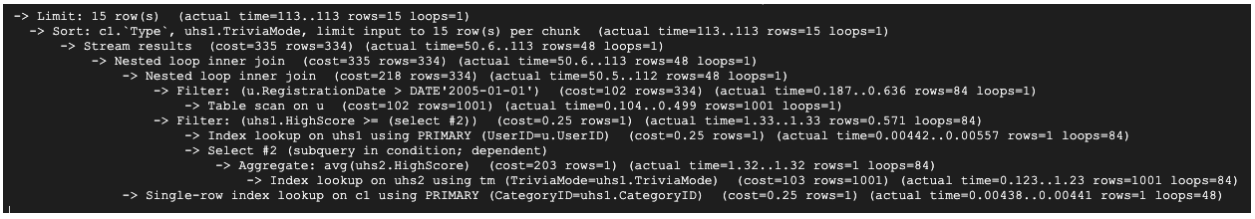
Total Cost = 335

Index on UserHighScore(HighScore) Performance:



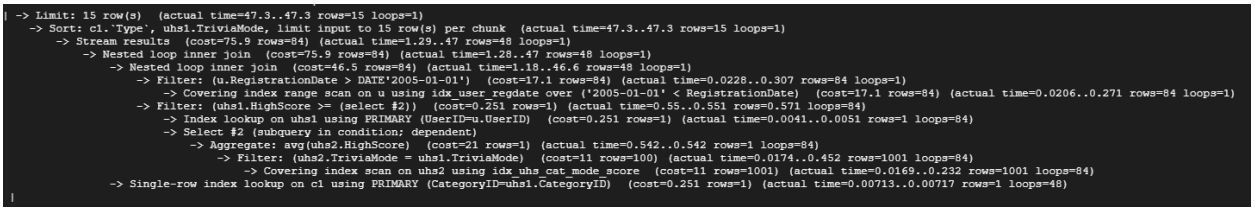
Total Cost = 335

Index on UserHighScore(TriviaMode) Performance:



Total Cost = 335

Index on User(RegistrationDate) Performance:



Total Cost = 75.9

Indexing Evaluation:
Indexing on the RegistrationDate attribute in the User table was most effective. This is because the WHERE clause filters on RegistrationDate, allowing the database to quickly eliminate a large

portion of users who registered before 2005, thus reducing the number of rows that need to be processed in the joins and subquery. The indexes on HighScore and TriviaMode didn't improve performance because these fields are used in comparisons (HighScore) or equality checks (TriviaMode) within a correlated subquery. The database still needs to calculate the average high score for each trivia mode, which requires scanning the entire UserHighScore table regardless of indexing.