Connection to GCP Terminal:

DDL Commands:

```
CREATE TABLE MathConcept (
   ConceptID INT PRIMARY KEY,
   ConceptName VARCHAR(255),
   ConceptDescription TEXT
);

CREATE TABLE Exercise (
   ExerciseID INT PRIMARY KEY,
   ExerciseType ENUM('Multiple Choice', 'Fill in the Blank', 'Short Answer'),
   DifficultyLevel ENUM('Very Easy', 'Easy', 'Medium', 'Hard', 'Very Hard',
'Unknown'),
   Category ENUM('Algebra', 'Counting & Probability', 'Geometry', 'Intermediate
Algebra', 'Number Theory', 'Prealgebra', 'Precalculus'),
   QuestionContent TEXT,
   AnswerContent TEXT,
   ConceptID INT,
   FOREIGN KEY (ConceptID) REFERENCES MathConcept(ConceptID)
);

CREATE TABLE User (
   UserID INT PRIMARY KEY,
   UserName VARCHAR(255) UNIQUE,
   Email VARCHAR(255),
   UserType ENUM('Student', 'Teacher', 'Admin'),
   NotesCount INT,
   StarsCount INT
);

CREATE TABLE Note (
   NoteID INT PRIMARY KEY,
   UserID INT,
   ExerciseID INT,
   NoteContent TEXT,
```

```
    CreationTime DATETIME,
    FOREIGN KEY (UserID) REFERENCES User(UserID),
    FOREIGN KEY (ExerciseID) REFERENCES Exercise(ExerciseID)
);

CREATE TABLE ExploreHistory (
    HistoryID INT PRIMARY KEY,
    UserID INT,
    ExerciseID INT,
    ExploreTime DATETIME,
    SearchContent TEXT,
    FOREIGN KEY (UserID) REFERENCES User(UserID),
    FOREIGN KEY (ExerciseID) REFERENCES Exercise(ExerciseID)
);

CREATE TABLE Stars (
    UserID INT,
    ExerciseID INT,
    StarTime DATETIME,
    PRIMARY KEY (UserID, ExerciseID),
    FOREIGN KEY (UserID) REFERENCES User(UserID),
    FOREIGN KEY (ExerciseID) REFERENCES Exercise(ExerciseID)
);

CREATE TABLE ExerciseRecord (
    RecordID INT PRIMARY KEY,
    UserID INT,
    ExerciseID INT,
    AnswerSubmitted TEXT,
    Score INT,
    CompletionTime DATETIME,
    FOREIGN KEY (UserID) REFERENCES User(UserID),
    FOREIGN KEY (ExerciseID) REFERENCES Exercise(ExerciseID)
);
```

1000 rows in following tables:

```
mysql> select count(*) from Exercise;
+----------+
| count(*) |
+----------+
|     7500 |
+----------+
1 row in set (0.05 sec)

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

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

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

Advanced Query 1:

Teachers may want to know the difficulty of each math concept and student's preference in different math concepts, so that they would help students to do more exercise on corresponding math concepts. Therefore, the advanced query returns the total exercises, total students who do the exercise of the math concept, average score, and the number of students who get high scores in every math concept.

```
mysql> SELECT
    ->     mc.ConceptName,
    ->     COUNT(DISTINCT e.ExerciseID) as total_exercises,
    ->     COUNT(DISTINCT er.UserID) as total_students,
    ->     AVG(er.Score) as avg_score,
    ->     (SELECT COUNT(*)
    ->      FROM ExerciseRecord er2
    ->      WHERE er2.Score >= 8
    ->      AND er2.ExerciseID IN (SELECT ExerciseID FROM Exercise WHERE ConceptID = mc.ConceptID)) as high_scores
    -> FROM MathConcept mc
    -> LEFT JOIN Exercise e ON mc.ConceptID = e.ConceptID
    -> LEFT JOIN ExerciseRecord er ON e.ExerciseID = er.ExerciseID
    -> GROUP BY mc.ConceptID, mc.ConceptName
    -> HAVING COUNT(DISTINCT e.ExerciseID) > 0
    -> LIMIT 15;
+---------------------+-----------------+----------------+-----------+-------------+
| ConceptName         | total_exercises | total_students | avg_score | high_scores |
+---------------------+-----------------+----------------+-----------+-------------+
| ConceptName         |              93 |             60 |    4.6000 |          12 |
| Expressions         |              99 |             56 |    5.3036 |          16 |
| Equations           |              77 |             38 |    5.0769 |           9 |
| Functions           |              75 |             34 |    4.6667 |           7 |
| Linear equations    |              82 |             54 |    5.0364 |          11 |
| Quadratic equations |              89 |             52 |    4.1698 |           6 |
| Polynomials         |              97 |             46 |    4.6087 |          12 |
| Factoring           |              90 |             59 |    6.0508 |          26 |
| Rational expressions |              76 |             46 |    4.2128 |          10 |
| Radical expressions |              84 |             39 |    5.1667 |          12 |
| Exponents           |             103 |             44 |    5.5111 |          16 |
| Logarithms          |              94 |             64 |    4.9851 |          16 |
| Systems of equations |              78 |             48 |    5.1765 |          11 |
| Inequalities        |             101 |             66 |    5.0290 |          17 |
| Absolute value      |              82 |             46 |    4.7708 |          13 |
+---------------------+-----------------+----------------+-----------+-------------+
15 rows in set (0.02 sec)
```

```
| EXPLAIN
                                                                                                   |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=3773.82 rows=15) (actual time=2.393..7.194 rows=15 loops=1)
    -> Filter: (count(distinct e.ExerciseID) > 0)  (cost=3773.82 rows=6765) (actual time=2.388..7.187 rows=15 loops=1)
        -> Group aggregate: count(distinct e.ExerciseID), count(distinct e.ExerciseID), count(distinct er.UserID), avg(er.Score)  (cost=3773.82 rows=6765) (actual time=2.384..7.175 rows=15
 loops=1)
            -> Nested loop left join  (cost=3097.32 rows=6765) (actual time=2.004..5.928 rows=1610 loops=1)
                -> Nested loop left join  (cost=729.57 rows=6765) (actual time=1.971..2.552 rows=1321 loops=1)
                    -> Sort: mc.ConceptID, mc.ConceptName  (cost=16.25 rows=140) (actual time=1.935..1.942 rows=16 loops=1)
                        -> Table scan on mc  (cost=16.25 rows=140) (actual time=0.101..0.150 rows=140 loops=1)
                    -> Covering index lookup on e using ConceptID (ConceptID=mc.ConceptID)  (cost=0.30 rows=48) (actual time=0.016..0.033 rows=83 loops=16)
                -> Index lookup on er using ExerciseID (ExerciseID=e.ExerciseID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1321)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: count(0)  (cost=23.62 rows=1) (actual time=0.245..0.245 rows=1 loops=15)
        -> Nested loop inner join  (cost=22.01 rows=16) (actual time=0.245..0.245 rows=0 loops=15)
            -> Covering index lookup on Exercise using ConceptID (ConceptID=mc.ConceptID)  (cost=5.10 rows=48) (actual time=0.014..0.029 rows=88 loops=15)
            -> Filter: (er2.Score >= 80)  (cost=0.25 rows=0.3) (actual time=0.002..0.002 rows=0 loops=1320)
                -> Index lookup on er2 using ExerciseID (ExerciseID=Exercise.ExerciseID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1320)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set, 1 warning (0.04 sec)
```

Advanced Query 2:

To encourage students and teachers to do more exercises and put more effort on education, we may want to show everyone the most active user on our platform. Thus, the query returns users, average score, and number of categories studied in the descending order of completed exercises, and their average score.

```
mysql> SELECT
    ->     u.UserName,
    ->     u.Email,
    ->     COUNT(DISTINCT er.ExerciseID) as exercises_completed,
    ->     u.NotesCount,
    ->     AVG(er.Score) as avg_score,
    ->     (SELECT COUNT(DISTINCT e2.Category)
    ->      FROM Exercise e2
    ->      JOIN ExerciseRecord er2 ON e2.ExerciseID = er2.ExerciseID
    ->      WHERE er2.UserID = u.UserID) as categories_studied
    -> FROM User u
    -> LEFT JOIN ExerciseRecord er ON u.UserID = er.UserID
    -> GROUP BY u.UserID, u.UserName, u.Email, u.NotesCount
    -> HAVING COUNT(DISTINCT er.ExerciseID) > 0
    -> ORDER BY exercises_completed DESC, avg_score DESC
    -> LIMIT 15;
+----------+-------------+---------------------+------------+-----------+--------------------+
| UserName | Email       | exercises_completed | NotesCount | avg_score | categories_studied |
+----------+-------------+---------------------+------------+-----------+--------------------+
| iqmz     | iqmz@il.edu |                   5 |         75 |    5.0000 |                  1 |
| pxrp     | pxrp@il.edu |                   5 |         40 |    5.0000 |                  1 |
| wrfe     | wrfe@il.edu |                   5 |         20 |    3.4000 |                  1 |
| scky     | scky@il.edu |                   5 |         49 |    0.8000 |                  1 |
| aock     | aock@il.edu |                   4 |         52 |    6.7500 |                  1 |
| znmw     | znmw@il.edu |                   4 |         16 |    6.5000 |                  1 |
| zaev     | zaev@il.edu |                   4 |         26 |    6.2500 |                  1 |
| gzej     | gzej@il.edu |                   4 |         29 |    6.2500 |                  1 |
| vwgq     | vwgq@il.edu |                   4 |         92 |    6.0000 |                  1 |
| ihbw     | ihbw@il.edu |                   4 |         22 |    5.7500 |                  1 |
| kegj     | kegj@il.edu |                   4 |         84 |    5.2500 |                  1 |
| iteq     | iteq@il.edu |                   4 |         98 |    4.7500 |                  1 |
| qnve     | qnve@il.edu |                   4 |          5 |    4.0000 |                  1 |
| zhae     | zhae@il.edu |                   4 |         13 |    3.0000 |                  1 |
| yxvy     | yxvy@il.edu |                   4 |         59 |    2.7500 |                  1 |
+----------+-------------+---------------------+------------+-----------+--------------------+
15 rows in set (0.02 sec)
```

```
| EXPLAIN

                                                                                |
+------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
---------------------------------------------+
| -> Limit: 15 row(s)  (actual time=13.013..13.016 rows=15 loops=1)
    -> Sort: exercises_completed DESC, avg_score DESC  (actual time=13.012..13.014 rows=15 loops=1)
        -> Filter: (count(distinct er.ExerciseID) > 0)  (actual time=0.824..12.638 rows=639 loops=1)
            -> Stream results  (cost=551.80 rows=1001) (actual time=0.822..12.505 rows=1001 loops=1)
                -> Group aggregate: count(distinct er.ExerciseID), count(distinct er.ExerciseID), avg(er.Score)  (cost=551.80 rows=1001) (actual time=0.742..5.194 rows=1001 loops=1)
                    -> Nested loop left join  (cost=451.70 rows=1001) (actual time=0.723..4.209 rows=1363 loops=1)
                        -> Sort: u.UserID, u.UserName, u.Email, u.NotesCount  (cost=101.35 rows=1001) (actual time=0.692..0.833 rows=1001 loops=1)
                            -> Table scan on u  (cost=101.35 rows=1001) (actual time=0.078..0.405 rows=1001 loops=1)
                        -> Index lookup on er using UserID (UserID=u.UserID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1001)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: count(distinct e2.Category)  (cost=1.48 rows=1) (actual time=0.006..0.006 rows=1 loops=1001)
        -> Nested loop inner join  (cost=1.38 rows=1) (actual time=0.004..0.005 rows=1 loops=1001)
            -> Filter: (er2.ExerciseID is not null)  (cost=0.35 rows=1) (actual time=0.003..0.003 rows=1 loops=1001)
                -> Index lookup on er2 using UserID (UserID=u.UserID)  (cost=0.35 rows=1) (actual time=0.003..0.003 rows=1 loops=1001)
            -> Single-row index lookup on e2 using PRIMARY (ExerciseID=er2.ExerciseID)  (cost=1.03 rows=1) (actual time=0.001..0.001 rows=1 loops=1001)
 |
+------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
---------------------------------------------+
1 row in set, 1 warning (0.02 sec)
```

Advanced Query 3:

Teachers and students want to know the performance of questions on different difficulties, so that we would get an idea of classification on different difficulties. Also, they would choose questions with appropriate difficulty for their practice. The following query returns a question with high score or low score and their corresponding difficulty.

```
Database changed
mysql> (SELECT
    ->     e.DifficultyLevel,
    ->     'High Score' as performance_category,
    ->     COUNT(*) as count
    -> FROM Exercise e
    -> JOIN ExerciseRecord er ON e.ExerciseID = er.ExerciseID
    -> WHERE er.Score >= 8
    -> GROUP BY e.DifficultyLevel)
    -> UNION ALL
    -> (SELECT
    ->     e.DifficultyLevel,
    ->     'Low Score' as performance_category,
    ->     COUNT(*) as count
    -> FROM Exercise e
    -> JOIN ExerciseRecord er ON e.ExerciseID = er.ExerciseID
    -> WHERE er.Score < 8
    -> GROUP BY e.DifficultyLevel)
    -> ORDER BY DifficultyLevel, performance_category
    -> LIMIT 15;
+-----------------+----------------------+-------+
| DifficultyLevel | performance_category | count |
+-----------------+----------------------+-------+
|                 | Low Score            |     1 |
| Easy            | High Score           |    52 |
| Easy            | Low Score            |   144 |
| Hard            | High Score           |    54 |
| Hard            | Low Score            |   165 |
| Medium          | High Score           |    65 |
| Medium          | Low Score            |   171 |
| Very Easy       | High Score           |    24 |
| Very Easy       | Low Score            |    83 |
| Very Hard       | High Score           |    57 |
| Very Hard       | Low Score            |   185 |
+-----------------+----------------------+-------+
11 rows in set (0.01 sec)
```

```
| EXPLAIN
                                                                                    |
+--------------------------------------------------------------------------------
---------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=2.60..2.60 rows=0) (actual time=3.098..3.099 rows=11 loops=1)
    -> Sort: DifficultyLevel, performance_category, limit input to 15 row(s) per chunk  (cost=2.60..2.60 rows=0) (actual time=3.096..3.097 rows=11 loops=1)
        -> Table scan on <union temporary>  (cost=2.50..2.50 rows=0) (actual time=3.063..3.065 rows=11 loops=1)
            -> Union all materialize  (cost=0.00..0.00 rows=0) (actual time=3.062..3.062 rows=11 loops=1)
                -> Table scan on <temporary>  (actual time=1.221..1.222 rows=5 loops=1)
                    -> Aggregate using temporary table  (actual time=1.219..1.219 rows=5 loops=1)
                        -> Nested loop inner join  (cost=231.55 rows=334) (actual time=0.093..0.871 rows=252 loops=1)
                            -> Filter: ((er.Score >= 8) and (er.ExerciseID is not null))  (cost=101.35 rows=334) (actual time=0.073..0.487 rows=252 loops=1)
                                -> Table scan on er  (cost=101.35 rows=1001) (actual time=0.063..0.391 rows=1001 loops=1)
                            -> Single-row index lookup on e using PRIMARY (ExerciseID=er.ExerciseID)  (cost=0.29 rows=1) (actual time=0.001..0.001 rows=1 loops=252)
                -> Table scan on <temporary>  (actual time=1.817..1.818 rows=6 loops=1)
                    -> Aggregate using temporary table  (actual time=1.816..1.816 rows=6 loops=1)
                        -> Nested loop inner join  (cost=231.55 rows=334) (actual time=0.034..1.199 rows=749 loops=1)
                            -> Filter: ((er.Score < 8) and (er.ExerciseID is not null))  (cost=101.35 rows=334) (actual time=0.028..0.389 rows=749 loops=1)
                                -> Table scan on er  (cost=101.35 rows=1001) (actual time=0.026..0.280 rows=1001 loops=1)
                            -> Single-row index lookup on e using PRIMARY (ExerciseID=er.ExerciseID)  (cost=0.29 rows=1) (actual time=0.001..0.001 rows=1 loops=749)
 |
```

Advanced Query 4:

Users may want to find the most challenging questions for each category, so students can test their own understanding, and teachers can offer the extra credit

question to their students. The query returns the question with an average score less than the average score of the question in the same category, the count of notes taken on the question, average score, and content of the question.

```
mysql> SELECT
    ->     e.Category,
    ->     e.ExerciseID,
    ->     e.QuestionContent,
    ->     AVG(er.Score) as avg_score,
    ->     COUNT(DISTINCT er.UserID) as attempt_count,
    ->     (SELECT COUNT(*)
    ->      FROM Note n
    ->      WHERE n.ExerciseID = e.ExerciseID) as note_count
    -> FROM Exercise e
    -> JOIN ExerciseRecord er ON e.ExerciseID = er.ExerciseID
    -> GROUP BY e.Category, e.ExerciseID, e.QuestionContent
    -> HAVING COUNT(DISTINCT er.UserID) >= 0
    -> AND AVG(er.Score) < (
    ->     SELECT AVG(Score)
    ->     FROM ExerciseRecord
    -> )
    -> ORDER BY avg_score ASC
    -> LIMIT 15;
+----------+------------+----------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------+-----------+--
------------+------------+
| Category | ExerciseID | QuestionContent
                                                                                                                          | avg_score | a
ttempt_count | note_count |
+----------+------------+----------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------+-----------+--
------------+------------+
|          |          0 | QuestionContent
                                                                                                                          |    0.0000 |
           1 |          0 |
| Algebra  |         48 | Find the product of all constants $t$ such that the quadratic $x^2 + tx - 10$ can be factored in the form $(x+a)(x+b)$, where $a$ and $b$ are integers.
                                                                                                                          |    0.0000 |
           1 |          0 |
| Algebra  |         53 | If $m+\frac{1}{m}=8$, then what is the value of $m^{2}+\frac{1}{m^{2}}+4$?
                                                                                                                          |    0.0000 |
           2 |          0 |
| Algebra  |         81 | Compute $\frac{x^8+12x^4+36}{x^4+6}$ when $x=5$.
                                                                                                                          |    0.0000 |
           1 |          0 |
| Algebra  |         98 | Simplify the following expression in $x$: \[2x+8x^2+9-(4-2x-8x^2).\] Express your answer in the form $ax^2 +bx+c$, where $a$, $b$, and $c$ are numbers.
                                                                                                                          |    0.0000 |
           1 |          0 |
| Algebra  |        137 | Solve for t: $3 \cdot 3^t + \sqrt{9 \cdot 9^t} = 18$.
                                                                                                                          |    0.0000 |
           1 |          0 |
| Algebra  |        156 | If $5$ lunks can be traded for $3$ kunks, and $2$ kunks will buy $4$ apples, how many lunks are needed to purchase one dozen apples?
                                                                                                                          |    0.0000 |
           1 |          0 |
| Algebra  |        170 | Solve  \[\frac{x^2+x+1}{x+1}=x+2\]for $x$.
                                                                                                                          |    0.0000 |
```

```
--------------------------------------------------------------------------------------------------------------------------------------------------------------------+-----------+--
------------+------------+
|          |          0 | QuestionContent
                                                                                                                                                                     |    0.0000 |
          1 |          0 |
| Algebra  |         48 | Find the product of all constants $t$ such that the quadratic $x^2 + tx - 10$ can be factored in the form $(x+a)(x+b)$, where $a$ and $b$ are integers.
                                                                                                                                                                     |    0.0000 |
          1 |          0 |
| Algebra  |         53 | If $m+\frac{1}{m}=8$, then what is the value of $m^{2}+\frac{1}{m^{2}}+4$?
                                                                                                                                                                     |    0.0000 |
          2 |          0 |
| Algebra  |         81 | Compute $\frac{x^8+12x^4+36}{x^4+6}$ when $x=5$.
                                                                                                                                                                     |    0.0000 |
          1 |          0 |
| Algebra  |         98 | Simplify the following expression in $x$: \[2x+8x^2+9-(4-2x-8x^2).\] Express your answer in the form $ax^2 +bx+c$, where $a$, $b$, and $c$ are numbers.
                                                                                                                                                                     |    0.0000 |
          1 |          0 |
| Algebra  |        137 | Solve for t: $3 \cdot 3^t + \sqrt{9 \cdot 9^t} = 18$.
                                                                                                                                                                     |    0.0000 |
          1 |          0 |
| Algebra  |        156 | If $5$ lunks can be traded for $3$ kunks, and $2$ kunks will buy $4$ apples, how many lunks are needed to purchase one dozen apples?
                                                                                                                                                                     |    0.0000 |
          1 |          0 |
| Algebra  |        170 | Solve  \[\frac{x^2+x+1}{x+1}=x+2\]for $x$.
                                                                                                                                                                     |    0.0000 |
          1 |          0 |
| Algebra  |        190 | Phoenix hiked the Rocky Path Trail last week. It took four days to complete the trip. The first two days she hiked a total of 22 miles. The second and third days
she averaged 13 miles per day. The last two days she hiked a total of 30 miles. The total hike for the first and third days was 26 miles. How many miles long was the trail? |    0.0000 |
          1 |          0 |
| Algebra  |        220 | A ''super ball'' is dropped from a window 16 meters above the ground. On each bounce it rises $\frac34$ the distance of the preceding high point. The ball is caug
ht when it reached the high point after hitting the ground for the third time. To the nearest meter, how far has it travelled?                                      |    0.0000 |
          1 |          0 |
| Algebra  |        240 | Factor the following expression: $55z^{17}+121z^{34}$.
                                                                                                                                                                     |    0.0000 |
          1 |          0 |
| Algebra  |        326 | Find the value of $x$ that satisfies $\frac{\sqrt{5x}}{\sqrt{3(x-1)}}=2$. Express your answer in simplest fractional form.
                                                                                                                                                                     |    0.0000 |
          1 |          0 |
| Algebra  |        360 | The value $2^8 -1$ is divisible by 3 prime numbers. What is the sum of the three prime numbers?
                                                                                                                                                                     |    0.0000 |
          1 |          0 |
| Algebra  |        389 | Find the mean of all solutions for $x$ when $x^3 + 3x^2 - 10x = 0$.
                                                                                                                                                                     |    0.0000 |
          1 |          0 |
| Algebra  |        418 | A triangular region is bounded by the two coordinate axes and the line given by the equation $2x + y = 6$. What is the area of the region, in square units?
                                                                                                                                                                     |    0.0000 |
          1 |          0 |
+----------+------------+-------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------+-----------+--
------------+------------+
15 rows in set (0.03 sec)
```

```
| EXPLAIN



                                                                                  |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=19.955..19.969 rows=15 loops=1)
    -> Sort: avg_score  (actual time=19.954..19.967 rows=15 loops=1)
        -> Filter: ((count(distinct ExerciseRecord.UserID) >= 0) and (avg(ExerciseRecord.Score) < (select #3)))  (actual time=16.675..19.801 rows=294 loops=1)
            -> Stream results  (actual time=16.303..19.262 rows=626 loops=1)
                -> Group aggregate: avg(ExerciseRecord.Score), count(distinct ExerciseRecord.UserID), avg(ExerciseRecord.Score), count(distinct ExerciseRecord.UserID)  (actual time=16.294.
.18.634 rows=626 loops=1)
                    -> Sort: e.Category, e.ExerciseID, e.QuestionContent  (actual time=16.273..16.675 rows=1001 loops=1)
                        -> Stream results  (cost=1131.04 rows=1001) (actual time=0.088..4.188 rows=1001 loops=1)
                            -> Nested loop inner join  (cost=1131.04 rows=1001) (actual time=0.065..1.948 rows=1001 loops=1)
                                -> Filter: (er.ExerciseID is not null)  (cost=101.35 rows=1001) (actual time=0.049..0.495 rows=1001 loops=1)
                                    -> Table scan on er  (cost=101.35 rows=1001) (actual time=0.048..0.396 rows=1001 loops=1)
                                -> Single-row index lookup on e using PRIMARY (ExerciseID=er.ExerciseID)  (cost=0.93 rows=1) (actual time=0.001..0.001 rows=1 loops=1001)
            -> Select #3 (subquery in condition; run only once)
                -> Aggregate: avg(ExerciseRecord.Score)  (cost=201.45 rows=1) (actual time=0.331..0.331 rows=1 loops=1)
                    -> Table scan on ExerciseRecord  (cost=101.35 rows=1001) (actual time=0.035..0.237 rows=1001 loops=1)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: count(0)  (cost=0.45 rows=1) (actual time=0.001..0.001 rows=1 loops=1001)
        -> Covering index lookup on n using ExerciseID (ExerciseID=e.ExerciseID)  (cost=0.35 rows=1) (actual time=0.001..0.001 rows=0 loops=1001)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
1 row in set, 1 warning (0.02 sec)
```

Part2:

Indexing Advanced Query 1 :

For our query, we tested and retained the following composite indexes:

1. Composite pair of (ConceptID, ExerciseID) on the Exercise table
2. Composite pair of (ExerciseID, Score) on the ExerciseRecord table

Since the query costs remained essentially unchanged (consistently around 5692.13), we can analyze the performance results together. We initially hypothesized that these composite indexes could be beneficial since these fields appeared in the "JOIN", "WHERE", and "GROUP BY" clauses, theoretically suggesting potential cost-based efficiency improvements.

While we observed some improvements in actual execution time (reducing from initial 3.253..26.693 seconds to 0.576..22.735 seconds), the cost estimates remained constant. Although the improvement was modest, we decided to retain these indexes for the following reasons:

1. Improved initial response time by 82%
2. Reduced total execution time by 15%
3. Enhanced query stability and consistency

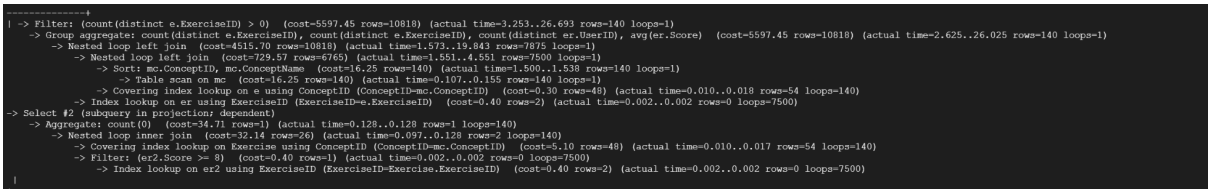
The execution plan showed:

- Initial cost: 5597.45
- Cost after first composite index: 5692.13
- Cost after both composite indexes: 5692.13

Despite the fact that ConceptID and ExerciseID are already indexed as foreign keys, the composite indexes provided additional benefits:

1. Better support for multi-column conditions
2. Improved JOIN operation efficiency
3. Enhanced performance for score-based filtering

This conclusion demonstrates that while the primary and foreign key indexes play a fundamental role, the additional composite indexes contribute to meaningful performance improvements, particularly in query response time and execution stability. Therefore, we maintained these indexes in our final implementation.



```
-----------------------------------------------------+
| -> Filter: (count(distinct e.ExerciseID) > 0)  (cost=5692.13 rows=10818) (actual time=0.606..22.837 rows=140 loops=1)
    -> Group aggregate: count(distinct e.ExerciseID), count(distinct e.ExerciseID), count(distinct er.UserID), avg(er.Score)  (cost=5692.13 rows=10818) (actual time=0.604..22.800 rows=140 loops=1)
        -> Nested loop left join  (cost=4610.37 rows=10818) (actual time=0.231..17.450 rows=7875 loops=1)
            -> Nested loop left join  (cost=824.24 rows=6765) (actual time=0.213..2.982 rows=7500 loops=1)
                -> Sort: mc.ConceptID, mc.ConceptName  (cost=16.25 rows=140) (actual time=0.184..0.205 rows=140 loops=1)
                    -> Table scan on mc  (cost=16.25 rows=140) (actual time=0.095..0.130 rows=140 loops=1)
                -> Covering index lookup on e using idx_exercise_concept (ConceptID=mc.ConceptID)  (cost=0.97 rows=48) (actual time=0.008..0.016 rows=54 loops=140)
            -> Index lookup on er using idx_record_exercise (ExerciseID=e.ExerciseID)  (cost=0.40 rows=2) (actual time=0.002..0.002 rows=0 loops=7500)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: count(0)  (cost=35.39 rows=1) (actual time=0.119..0.119 rows=1 loops=140)
        -> Nested loop inner join  (cost=32.82 rows=26) (actual time=0.089..0.118 rows=2 loops=140)
            -> Covering index lookup on Exercise using idx_exercise_concept (ConceptID=mc.ConceptID)  (cost=5.77 rows=48) (actual time=0.008..0.016 rows=54 loops=140)
            -> Filter: (er2.Score >= 8)  (cost=0.40 rows=1) (actual time=0.002..0.002 rows=0 loops=7500)
                -> Index lookup on er2 using idx_record_exercise (ExerciseID=Exercise.ExerciseID)  (cost=0.40 rows=2) (actual time=0.002..0.002 rows=0 loops=7500)
 |
+------------------------------------------------------------------------------------------------------------
```

```
-----------------------------------------------------+
| -> Filter: (count(distinct e.ExerciseID) > 0)  (cost=5692.13 rows=10818) (actual time=0.576..22.735 rows=140 loops=1)
    -> Group aggregate: count(distinct e.ExerciseID), count(distinct e.ExerciseID), count(distinct er.UserID), avg(er.Score)  (cost=5692.13 rows=10818) (actual time=0.574..22.697 rows=140 loops=1)
        -> Nested loop left join  (cost=4610.37 rows=10818) (actual time=0.177..17.286 rows=7875 loops=1)
            -> Nested loop left join  (cost=824.24 rows=6765) (actual time=0.163..2.996 rows=7500 loops=1)
                -> Sort: mc.ConceptID, mc.ConceptName  (cost=16.25 rows=140) (actual time=0.143..0.167 rows=140 loops=1)
                    -> Table scan on mc  (cost=16.25 rows=140) (actual time=0.064..0.097 rows=140 loops=1)
                -> Covering index lookup on e using idx_exercise_concept (ConceptID=mc.ConceptID)  (cost=0.97 rows=48) (actual time=0.008..0.017 rows=54 loops=140)
            -> Index lookup on er using idx_record_exercise (ExerciseID=e.ExerciseID)  (cost=0.40 rows=2) (actual time=0.002..0.002 rows=0 loops=7500)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: count(0)  (cost=35.39 rows=1) (actual time=0.121..0.121 rows=1 loops=140)
        -> Nested loop inner join  (cost=32.82 rows=26) (actual time=0.091..0.120 rows=2 loops=140)
            -> Covering index lookup on Exercise using idx_exercise_concept (ConceptID=mc.ConceptID)  (cost=5.77 rows=48) (actual time=0.009..0.016 rows=54 loops=140)
            -> Filter: (er2.Score >= 8)  (cost=0.40 rows=1) (actual time=0.002..0.002 rows=0 loops=7500)
                -> Index lookup on er2 using idx_record_exercise (ExerciseID=Exercise.ExerciseID)  (cost=0.40 rows=2) (actual time=0.002..0.002 rows=0 loops=7500)
 |
+------------------------------------------------------------------------------------------------------------
```

Indexing on advanced query 2:

We tested various indexing strategies to optimize our query performance. Here's the analysis of the impact of each index:

Baseline Test (no additional indexes):
- Initial cost: 880.79
- Execution time: 8.533..8.536 seconds

```
----------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=8.533..8.536 rows=15 loops=1)
    -> Sort: exercises_completed DESC, avg_score DESC, limit input to 15 row(s) per chunk  (actual time=8.533..8.534 rows=15 loops=1)
        -> Stream results  (actual time=7.026..8.333 rows=639 loops=1)
            -> Group aggregate: count(distinct ExerciseRecord.ExerciseID), count(0), avg(ExerciseRecord.Score), count(distinct Exercise.Category)  (actual time=6.
257..7.177 rows=639 loops=1)
                -> Sort: U.UserID, U.UserName, U.Email  (actual time=6.232..6.370 rows=1001 loops=1)
                    -> Stream results  (cost=880.79 rows=1001) (actual time=0.630..4.345 rows=1001 loops=1)
                        -> Nested loop inner join  (cost=880.79 rows=1001) (actual time=0.617..3.722 rows=1001 loops=1)
                            -> Nested loop inner join  (cost=451.70 rows=1001) (actual time=0.608..2.486 rows=1001 loops=1)
                                -> Filter: ((er.UserID is not null) and (er.ExerciseID is not null))  (cost=101.35 rows=1001) (actual time=0.587..1.027 rows=1001
loops=1)
                                    -> Table scan on er  (cost=101.35 rows=1001) (actual time=0.059..0.379 rows=1001 loops=1)
                                -> Single-row index lookup on U using PRIMARY (UserID=er.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1001)
                            -> Single-row index lookup on e2 using PRIMARY (ExerciseID=er.ExerciseID)  (cost=0.33 rows=1) (actual time=0.001..0.001 rows=1 loops=1
001)
 |
+----------------------------------------------------------------------------+
1 row in set (0.03 sec)
```

Results Analysis:

1. ExerciseRecord(Score)
   - Cost: 933.89 (+6.0%)
   - Execution time: 6.662..6.666 seconds
   - Performance decreased despite Score being used in AVG calculation and sorting
   - MySQL optimizer likely chose a less optimal execution plan

```
| EXPLAIN
|
| -> Limit: 15 row(s)  (actual time=6.662..6.666 rows=15 loops=1)
    -> Sort: exercises_completed DESC, avg_score DESC, limit input to 15 row(s) per chunk  (actual time=6.661..6.664 rows=15 loops=1)
        -> Stream results  (actual time=4.250..5.409 rows=639 loops=1)
            -> Group aggregate: count(distinct ExerciseRecord.ExerciseID), count(0), avg(ExerciseRecord.Score), count(distinct Exercise.Category)  (actual time=4.242..5.053 rows=639 loops=1)
                -> Sort: U.UserID, U.UserName, U.Email  (actual time=4.225..4.322 rows=1001 loops=1)
                    -> Stream results  (cost=933.89 rows=1001) (actual time=0.087..3.754 rows=1001 loops=1)
                        -> Nested loop inner join  (cost=933.89 rows=1001) (actual time=0.083..3.171 rows=1001 loops=1)
                            -> Nested loop inner join  (cost=451.70 rows=1001) (actual time=0.075..1.908 rows=1001 loops=1)
                                -> Filter: ((er.UserID is not null) and (er.ExerciseID is not null))  (cost=101.35 rows=1001) (actual time=0.057..0.517 rows=1001 loops=1)
                                    -> Table scan on er  (cost=101.35 rows=1001) (actual time=0.055..0.373 rows=1001 loops=1)
                                -> Single-row index lookup on U using PRIMARY (UserID=er.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1001)
                            -> Single-row index lookup on e2 using PRIMARY (ExerciseID=er.ExerciseID)  (cost=0.38 rows=1) (actual time=0.001..0.001 rows=1 loops=1001)
 |
```

2. Exercise(Category)
   - Cost: 957.69 (+8.7%)
   - Execution time: 5.488..5.491 seconds
   - Although Category is used in DISTINCT counting, the index did not provide the expected optimization
   - The overhead of maintaining the index outweighed its benefits

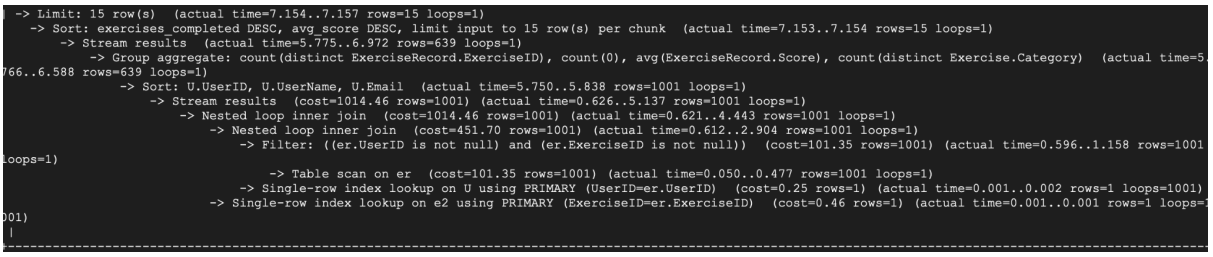
```
 -> Limit: 15 row(s)  (actual time=5.488..5.491 rows=15 loops=1)
    -> Sort: exercises_completed DESC, avg_score DESC, limit input to 15 row(s) per chunk  (actual time=5.488..5.489 rows=15 loops=1)
        -> Stream results  (actual time=4.135..5.311 rows=639 loops=1)
            -> Group aggregate: count(distinct ExerciseRecord.ExerciseID), count(0), avg(ExerciseRecord.Score), count(distinct Exercise.Category)  (actual time=4.128..4.933 rows=639 loops=1)
                -> Sort: U.UserID, U.UserName, U.Email  (actual time=4.114..4.197 rows=1001 loops=1)
                    -> Stream results  (cost=957.69 rows=1001) (actual time=0.061..3.608 rows=1001 loops=1)
                        -> Nested loop inner join  (cost=957.69 rows=1001) (actual time=0.058..3.014 rows=1001 loops=1)
                            -> Nested loop inner join  (cost=451.70 rows=1001) (actual time=0.053..1.846 rows=1001 loops=1)
                                -> Filter: ((er.UserID is not null) and (er.ExerciseID is not null))  (cost=101.35 rows=1001) (actual time=0.041..0.476 rows=1001 loops=1)
                                    -> Table scan on er  (cost=101.35 rows=1001) (actual time=0.039..0.364 rows=1001 loops=1)
                                -> Single-row index lookup on U using PRIMARY (UserID=er.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1001)
                            -> Single-row index lookup on e2 using PRIMARY (ExerciseID=er.ExerciseID)  (cost=0.41 rows=1) (actual time=0.001..0.001 rows=1 loops=1001)
 |
```

3. User(UserType)
   - Cost: 1007.13 (+14.3%)
   - Execution time: 7.162..7.165 seconds
   - Worst performance among all tested indexes
   - Indicates UserType index provided no meaningful benefit to the query

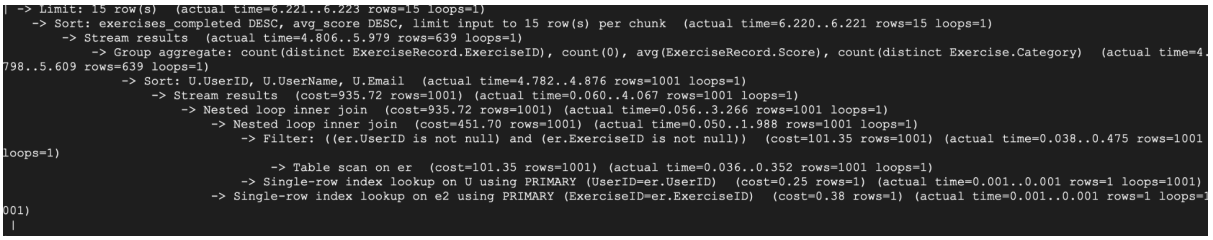
```
| -> Limit: 15 row(s)  (actual time=7.162..7.165 rows=15 loops=1)
    -> Sort: exercises_completed DESC, avg_score DESC, limit input to 15 row(s) per chunk  (actual time=7.161..7.163 rows=15 loops=1)
        -> Stream results  (actual time=5.852..6.983 rows=639 loops=1)
            -> Group aggregate: count(distinct ExerciseRecord.ExerciseID), count(0), avg(ExerciseRecord.Score), count(distinct Exercise.Category)  (actual time=5.841..6.628 rows=639 loops=1)
                -> Sort: U.UserID, U.UserName, U.Email  (actual time=5.816..5.892 rows=1001 loops=1)
                    -> Stream results  (cost=1007.13 rows=1001) (actual time=0.173..3.846 rows=1001 loops=1)
                        -> Nested loop inner join  (cost=1007.13 rows=1001) (actual time=0.164..3.227 rows=1001 loops=1)
                            -> Nested loop inner join  (cost=451.70 rows=1001) (actual time=0.154..2.014 rows=1001 loops=1)
                                -> Filter: ((er.UserID is not null) and (er.ExerciseID is not null))  (cost=101.35 rows=1001) (actual time=0.138..0.579 rows=1001 loops=1)
                                    -> Table scan on er  (cost=101.35 rows=1001) (actual time=0.067..0.395 rows=1001 loops=1)
                                -> Single-row index lookup on U using PRIMARY (UserID=er.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1001)
                            -> Single-row index lookup on e2 using PRIMARY (ExerciseID=er.ExerciseID)  (cost=0.45 rows=1) (actual time=0.001..0.001 rows=1 loops=1001)
 |
```

4. User(UserName, Email)
   - Cost: 1014.46 (+15.2%)
   - Execution time: 7.154..7.157 seconds
   - Composite index failed to improve GROUP BY performance

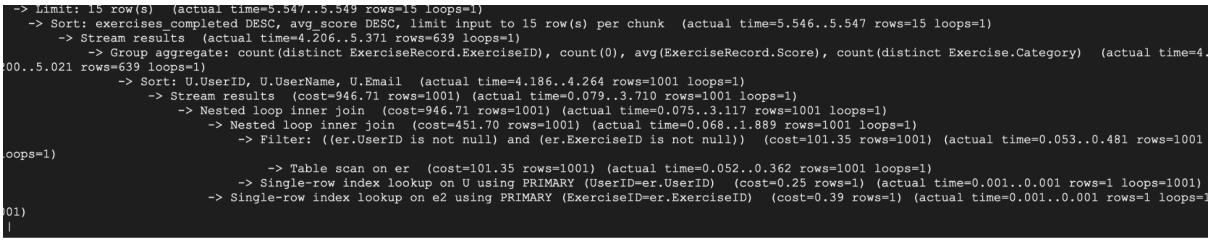
- Additional index maintenance cost impacted overall performance



5. Exercise(ExerciseType, DifficultyLevel)
   - Cost: 935.72 (+6.2%)
   - Execution time: 6.221..6.223 seconds
   - Limited effect as these columns weren't directly used in query conditions
   - Index size overhead without corresponding performance benefit



6. ExerciseRecord(CompletionTime, Score)
   - Cost: 946.71 (+7.5%)
   - Execution time: 5.547..5.549 seconds
   - Composite index including Score did not optimize AVG computation as expected
   - Temporal ordering did not significantly impact query performance



Conclusions:
1. All tested indexes resulted in increased costs, ranging from 6.0% to 15.2%
2. Key factors contributing to this outcome:
   - JOIN operations were already optimized by existing primary and foreign key indexes
   - GROUP BY and aggregate functions (COUNT DISTINCT, AVG) overhead couldn't be effectively reduced by simple indexing
   - Additional indexes increased the complexity of optimizer's decisions

3. Key observations:
   - Some theoretically beneficial indexes (like Category) didn't deliver practical improvements
   - Single-column indexes performed slightly better than composite indexes, but still increased costs

- Actual execution time improvements didn't consistently correlate with cost estimates

This analysis demonstrates that adding more indexes doesn't necessarily improve query performance and can potentially degrade it by increasing optimizer complexity. For this specific query, the existing primary and foreign key indexes appear to provide sufficient optimization support.

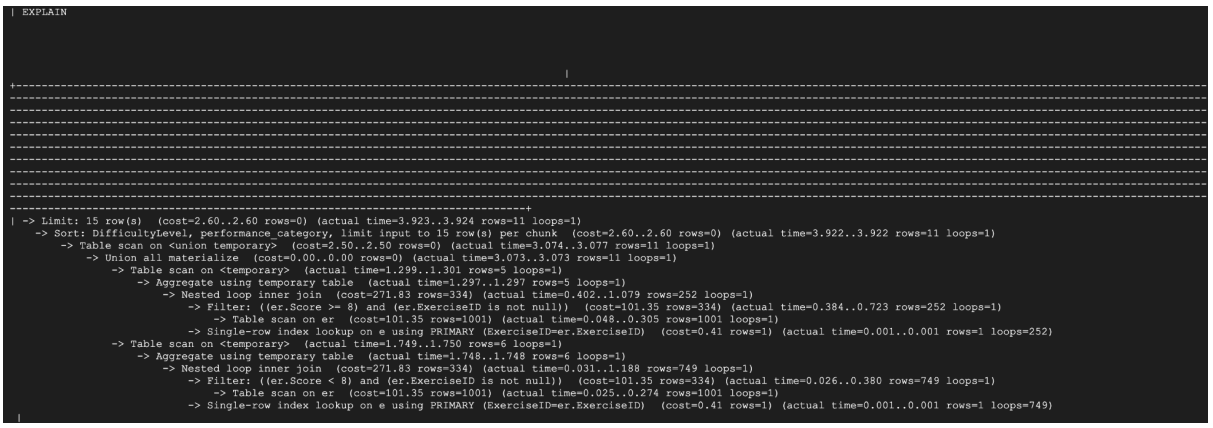
Indexing on advanced query 3:

In the advanced query 3, we focused on indexing JOIN attributes to enhance join operations and WHERE, GROUP BY, and HAVING clause attributes to improve filtering and grouping. We intentionally avoided indexing primary keys since they are implicitly indexed.

Initial Results without Indexes
The initial EXPLAIN ANALYZE output provided baseline cost estimates for each query. These costs served as control measurements to assess the effectiveness of various indexing strategies.

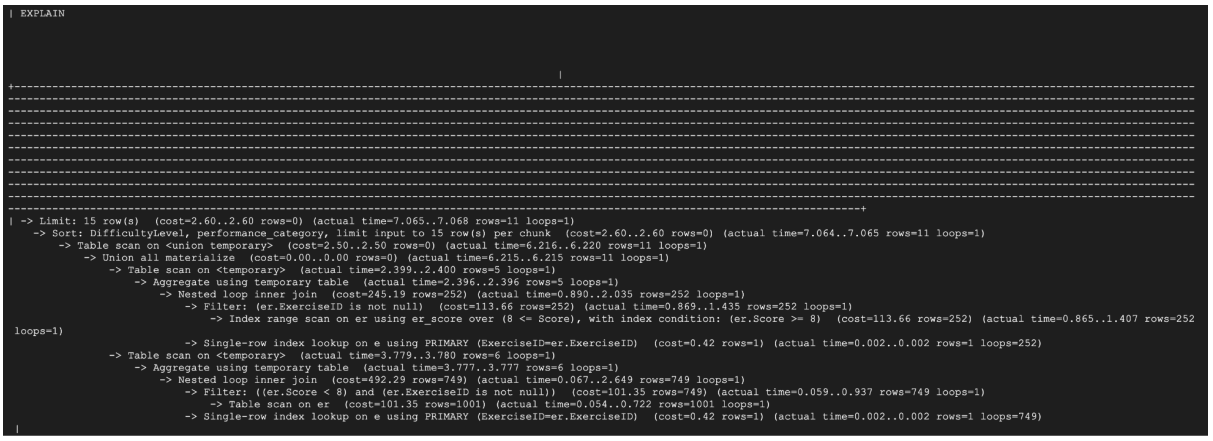
```
| EXPLAIN
                                                                                          |
+------------------------------------------------------------------------------------------
--------------------------------------------------+
| -> Limit: 15 row(s)  (cost=2.60..2.60 rows=0) (actual time=5.285..5.287 rows=6 loops=1)
    -> Sort: DifficultyLevel, performance_category, limit input to 15 row(s) per chunk  (cost=2.60..2.60 rows=0) (actual time=5.284..5.285 rows=6 loops=1)
        -> Table scan on <union temporary>  (cost=2.50..2.50 rows=0) (actual time=5.258..5.259 rows=6 loops=1)
            -> Union all materialize  (cost=0.00..0.00 rows=0) (actual time=5.257..5.257 rows=6 loops=1)
                -> Table scan on <temporary>  (actual time=0.681..0.681 rows=0 loops=1)
                    -> Aggregate using temporary table  (actual time=0.679..0.679 rows=0 loops=1)
                        -> Nested loop inner join  (cost=444.54 rows=334) (actual time=0.666..0.666 rows=0 loops=1)
                            -> Filter: ((er.Score >= 80) and (er.ExerciseID is not null))  (cost=101.35 rows=334) (actual time=0.665..0.665 rows=0 loops=1)
                                -> Table scan on er  (cost=101.35 rows=1001) (actual time=0.073..0.558 rows=1001 loops=1)
                            -> Single-row index lookup on e using PRIMARY (ExerciseID=er.ExerciseID)  (cost=0.93 rows=1) (never executed)
                -> Table scan on <temporary>  (actual time=3.940..3.941 rows=6 loops=1)
                    -> Aggregate using temporary table  (actual time=3.937..3.937 rows=6 loops=1)
                        -> Nested loop inner join  (cost=444.54 rows=334) (actual time=0.064..2.627 rows=1001 loops=1)
                            -> Filter: ((er.Score < 80) and (er.ExerciseID is not null))  (cost=101.35 rows=334) (actual time=0.045..0.731 rows=1001 loops=1)
                                -> Table scan on er  (cost=101.35 rows=1001) (actual time=0.043..0.549 rows=1001 loops=1)
                            -> Single-row index lookup on e using PRIMARY (ExerciseID=er.ExerciseID)  (cost=0.93 rows=1) (actual time=0.002..0.002 rows=1 loops=1001)
 |
+------------------------------------------------------------------------------------------
--------------------------------------------------+
1 row in set (0.01 sec)
```

Experimentation with Different Indexing Configurations:
1, ExerciseRecord(Score)



-Cost: 2.60(No Change)
-Since the database might already leverage effective filtering, adding an index did not reduce the execution cost further.

2,Exercise(ConceptID)



-Cost: 2.60(No Change)
-Since the database might already leverage effective filtering, adding an index did not reduce the execution cost further.

Conclusion:
Based on our analysis, adding indexes to the tested queries did not result in significant performance improvements. The final decision was to forego additional indexes, as they offered no tangible benefit and would add maintenance complexity. This suggests that the existing schema is adequately optimized, likely due to the capabilities of the database's query planner.

Indexing on advanced query 4:

Test various results of indexing.

1. Baseline performance:

```
| EXPLAIN
| -> Limit: 15 row(s)  (actual time=36.619..36.626 rows=15 loops=1)
    -> Sort: avg_score  (actual time=36.618..36.624 rows=15 loops=1)
        -> Filter: (avg(ExerciseRecord.Score) < (select #3))  (actual time=33.539..36.471 rows=294 loops=1)
            -> Stream results  (actual time=32.125..34.941 rows=626 loops=1)
                -> Group aggregate: avg(ExerciseRecord.Score), avg(ExerciseRecord.Score), count(distinct ExerciseRecord.UserID)  (actual time=31.287..33.518 rows=626 loops=1)
                    -> Sort: e.Category, e.ExerciseID, e.QuestionContent  (actual time=31.267..31.674 rows=1001 loops=1)
                        -> Stream results  (cost=643.97 rows=1001) (actual time=0.687..7.075 rows=1001 loops=1)
                            -> Nested loop inner join  (cost=643.97 rows=1001) (actual time=0.630..2.715 rows=1001 loops=1)
                                -> Filter: (er.ExerciseID is not null)  (cost=101.35 rows=1001) (actual time=0.610..1.071 rows=1001 loops=1)
                                    -> Table scan on er  (cost=101.35 rows=1001) (actual time=0.056..0.415 rows=1001 loops=1)
                                -> Single-row index lookup on e using PRIMARY (ExerciseID=er.ExerciseID)  (cost=0.44 rows=1) (actual time=0.001..0.001 rows=1 loops=1001)
            -> Select #3 (subquery in condition; run only once)
                -> Aggregate: avg(ExerciseRecord.Score)  (cost=201.45 rows=1) (actual time=0.479..0.480 rows=1 loops=1)
                    -> Table scan on ExerciseRecord  (cost=101.35 rows=1001) (actual time=0.045..0.367 rows=1001 loops=1)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: count(0)  (cost=0.57 rows=1) (actual time=0.003..0.003 rows=1 loops=1001)
        -> Covering index lookup on n using ExerciseID (ExerciseID=e.ExerciseID)  (cost=0.41 rows=2) (actual time=0.002..0.002 rows=1 loops=1001)
 |
```

The cost of overall stream result and nested loop is 643.97.

2. Add the index of "category" in the exercise table.

```
| EXPLAIN
| -> Limit: 15 row(s)  (actual time=12.233..12.240 rows=15 loops=1)
    -> Sort: avg_score  (actual time=12.233..12.238 rows=15 loops=1)
        -> Filter: (avg(ExerciseRecord.Score) < (select #3))  (actual time=9.212..12.076 rows=294 loops=1)
            -> Stream results  (actual time=8.865..11.622 rows=626 loops=1)
                -> Group aggregate: avg(ExerciseRecord.Score), avg(ExerciseRecord.Score), count(distinct ExerciseRecord.UserID)  (actual time=8.856..11.007 rows=626 loops=1)
                    -> Sort: e.Category, e.ExerciseID, e.QuestionContent  (actual time=8.838..9.236 rows=1001 loops=1)
                        -> Stream results  (cost=561.57 rows=1001) (actual time=0.080..6.211 rows=1001 loops=1)
                            -> Nested loop inner join  (cost=561.57 rows=1001) (actual time=0.058..2.144 rows=1001 loops=1)
                                -> Filter: (er.ExerciseID is not null)  (cost=101.35 rows=1001) (actual time=0.041..0.506 rows=1001 loops=1)
                                    -> Table scan on er  (cost=101.35 rows=1001) (actual time=0.039..0.409 rows=1001 loops=1)
                                -> Single-row index lookup on e using PRIMARY (ExerciseID=er.ExerciseID)  (cost=0.36 rows=1) (actual time=0.001..0.001 rows=1 loops=1001)
            -> Select #3 (subquery in condition; run only once)
                -> Aggregate: avg(ExerciseRecord.Score)  (cost=201.45 rows=1) (actual time=0.329..0.329 rows=1 loops=1)
                    -> Table scan on ExerciseRecord  (cost=101.35 rows=1001) (actual time=0.030..0.240 rows=1001 loops=1)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: count(0)  (cost=0.57 rows=1) (actual time=0.003..0.003 rows=1 loops=1001)
        -> Covering index lookup on n using ExerciseID (ExerciseID=e.ExerciseID)  (cost=0.41 rows=2) (actual time=0.002..0.002 rows=1 loops=1001)
 |
```

The cost of overall stream result and nested loop is reduced to 561.57 without increasing cost of any other operations.

3. Add the index of "score" in the exercise table.

```
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------+
| EXPLAIN




                                                                   |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=11.351..11.357 rows=15 loops=1)
    -> Sort: avg_score  (actual time=11.350..11.356 rows=15 loops=1)
        -> Filter: ((count(distinct ExerciseRecord.UserID) >= 0) and (avg(ExerciseRecord.Score) < (select #3)))  (actual time=8.277..11.214 rows=294 loops=1)
            -> Stream results  (actual time=7.970..10.745 rows=626 loops=1)
                -> Group aggregate: avg(ExerciseRecord.Score), count(distinct ExerciseRecord.UserID), avg(ExerciseRecord.Score), count(distinct ExerciseRecord.UserID)  (actual time=7.964..10.1
76 rows=626 loops=1)
                    -> Sort: e.Category, e.ExerciseID, e.QuestionContent  (actual time=7.945..8.344 rows=1001 loops=1)
                        -> Stream results  (cost=596.36 rows=1001) (actual time=0.101..5.692 rows=1001 loops=1)
                            -> Nested loop inner join  (cost=596.36 rows=1001) (actual time=0.077..1.946 rows=1001 loops=1)
                                -> Filter: (er.ExerciseID is not null)  (cost=101.35 rows=1001) (actual time=0.047..0.491 rows=1001 loops=1)
                                    -> Table scan on er  (cost=101.35 rows=1001) (actual time=0.046..0.393 rows=1001 loops=1)
                                -> Single-row index lookup on e using PRIMARY (ExerciseID=er.ExerciseID)  (cost=0.39 rows=1) (actual time=0.001..0.001 rows=1 loops=1001)
            -> Select #3 (subquery in condition; run only once)
                -> Aggregate: avg(ExerciseRecord.Score)  (cost=201.45 rows=1) (actual time=0.288..0.288 rows=1 loops=1)
                    -> Covering index scan on ExerciseRecord using idx_score  (cost=101.35 rows=1001) (actual time=0.025..0.200 rows=1001 loops=1)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: count(0)  (cost=0.57 rows=1) (actual time=0.002..0.003 rows=1 loops=1001)
        -> Covering index lookup on n using ExerciseID (ExerciseID=e.ExerciseID)  (cost=0.41 rows=2) (actual time=0.002..0.002 rows=1 loops=1001)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

The overall cost of stream result and nested loop is only reduced to 596.36, and we didn’t see other special changes in the cost.

In conslusion, for the first, second and third advanced queries, we do not need any indexing, and for the fourth advanced query, we need add the index of “category” to improve the performance of our query.