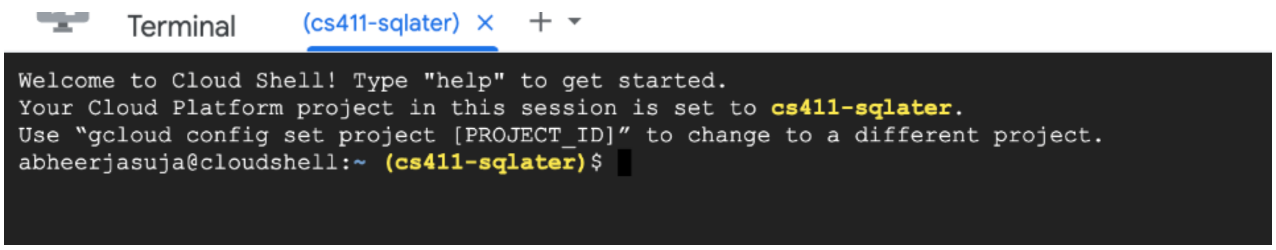
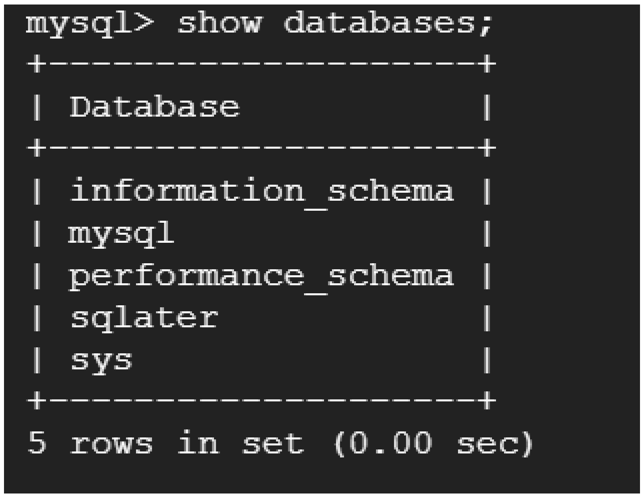
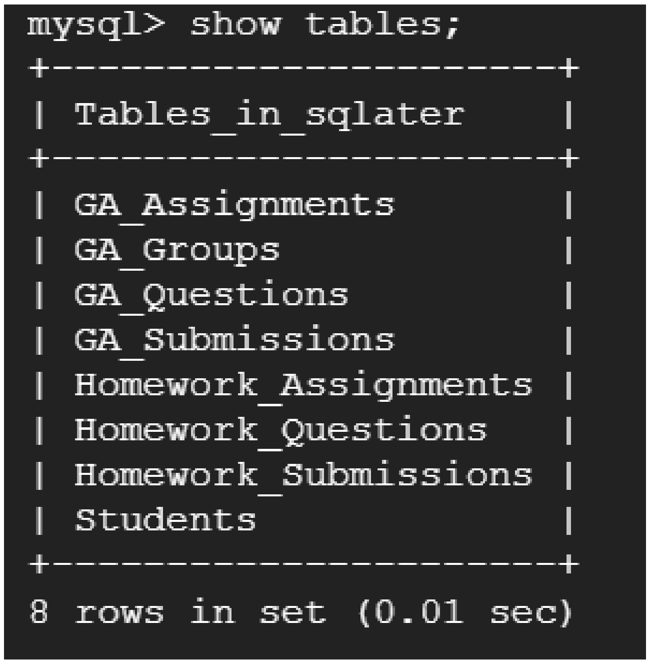
2.1 ScreenShot for Connection:



```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to cs411-sqlater.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
abheerjasuja@cloudshell:~ (cs411-sqlater)$
```



```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sqlater            |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

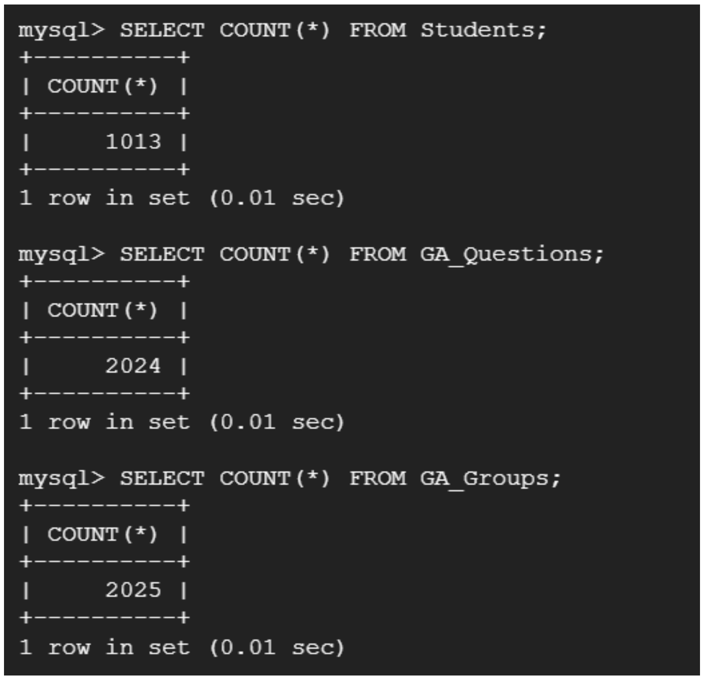


```
mysql> show tables;
+-----------------------+
| Tables_in_sqlater     |
+-----------------------+
| GA_Assignments        |
| GA_Groups             |
| GA_Questions          |
| GA_Submissions        |
| Homework_Assignments  |
| Homework_Questions    |
| Homework_Submissions  |
| Students              |
+-----------------------+
8 rows in set (0.01 sec)
```

2.2 DDL Commands to create the Tables:

```
CREATE TABLE Students (
      uin INT PRIMARY KEY,
      net_id VARCHAR(10),
      name VARCHAR(50)
);

CREATE TABLE Homework_Assignments (
      hw_id INT PRIMARY KEY,
      hw_name VARCHAR(50)
);

CREATE TABLE Homework_Questions (
      uin INT FOREIGN KEY REFERENCES Students(uin),
      hw_id INT FOREIGN KEY REFERENCES Homework_Assignments(hw_id),
      question_number INT,
      submissions INT,
      ungradable_submissions INT,
      correct_submissions INT,
      question_score REAL,
      PRIMARY KEY(question_number, hw_id, uin)
);


CREATE TABLE Homework_Submissions (
      uin INT FOREIGN KEY REFERENCES Students(uin),
      hw_id INT FOREIGN KEY REFERENCES Homework_Assignments(hw_id),
      time_taken INT,
      score INT,
      PRIMARY KEY(hw_id, UIN)
);


CREATE TABLE GA_Groups (
      uin INT FOREIGN KEY REFERENCES Students(uin),
      group_id INT
      );


CREATE TABLE GA_Submissions (
      group_id INT FOREIGN KEY REFERENCES GA_Groups(group_id),
      ga_id INT FOREIGN KEY REFERENCES GA_Assignments(ga_id),
      group_name VARCHAR(20),
      col_rating INT,
      time_taken INT,
      score INT,
      PRIMARY KEY (group_id, ga_id)
);



CREATE TABLE GA_Assignments (
      ga_id INT PRIMARY KEY,
      ga_name VARCHAR(20)
);

CREATE TABLE GA_Questions (
      group_id INT FOREIGN KEY REFERENCES GA_Groups(group_id),
      ga_id INT FOREIGN KEY REFERENCES GA_Assignments(ga_id),
      question_number INT,
      submissions INT,
      ungradable_submissions INT,
      correct_submissions INT,
      question_score REAL,
      PRIMARY KEY(question_number, group_id, ga_id)
);
```

2.3 Tables with at least 1000 rows



```
mysql> SELECT COUNT(*) FROM Students;
+----------+
| COUNT(*) |
+----------+
|     1013 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM GA_Questions;
+----------+
| COUNT(*) |
+----------+
|     2024 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM GA_Groups;
+----------+
| COUNT(*) |
+----------+
|     2025 |
+----------+
1 row in set (0.01 sec)
```

3 TWO Advanced SQL queries:

3.1 Design of two advanced queries

1) This SQL query gets the average score by question for every GA and homework assignment (one of ga_id and hw_id will be null, depending on the type of assignment)

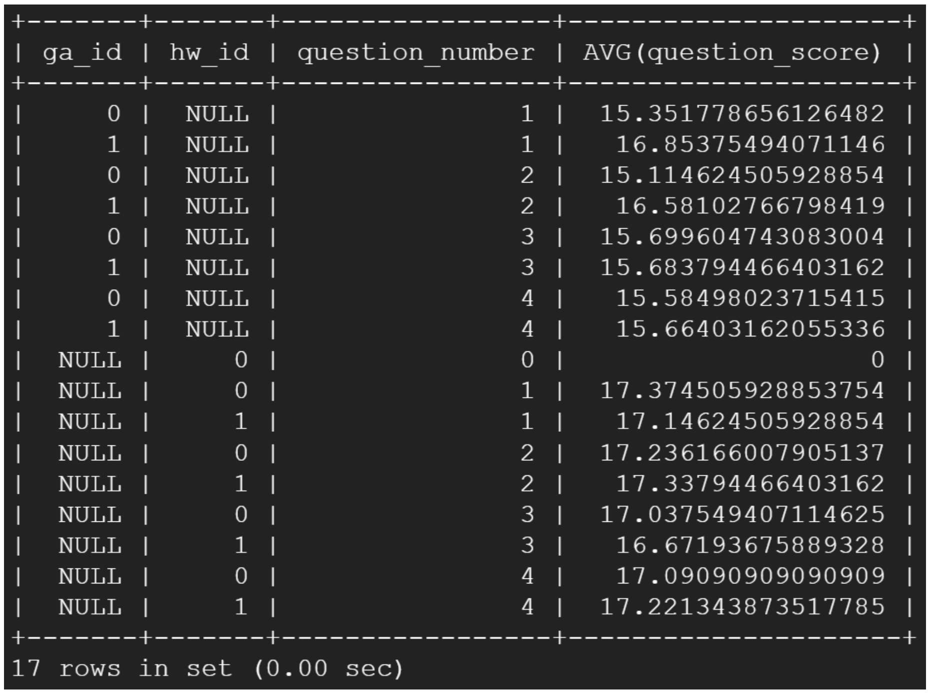
```
SELECT ga_id, NULL AS hw_id, question_number, AVG(question_score)
FROM GA_Questions
GROUP BY ga_id, question_number

UNION

SELECT NULL, hw_id, question_number, AVG(question_score)
FROM Homework_Questions
GROUP BY hw_id, question_number
```

2) UIN of Students who have an average GA_score across all GA assignments greater than 80.

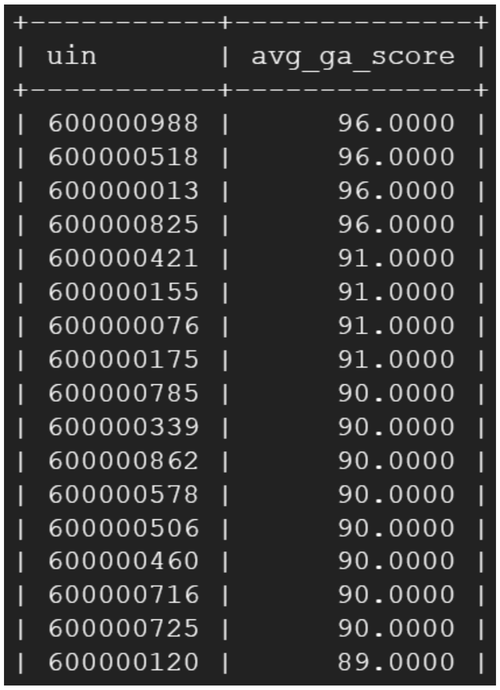
```
SELECT uin, AVG(score) avg_ga_score
FROM GA_Submissions NATURAL JOIN GA_Groups
WHERE ga_id IN (0,1)
GROUP BY uin
HAVING avg_ga_score >= 80
ORDER BY avg_ga_score DESC;
```

3.2 Screenshot of the top 15 rows of the SQL queries

Results from the first SQL Query.

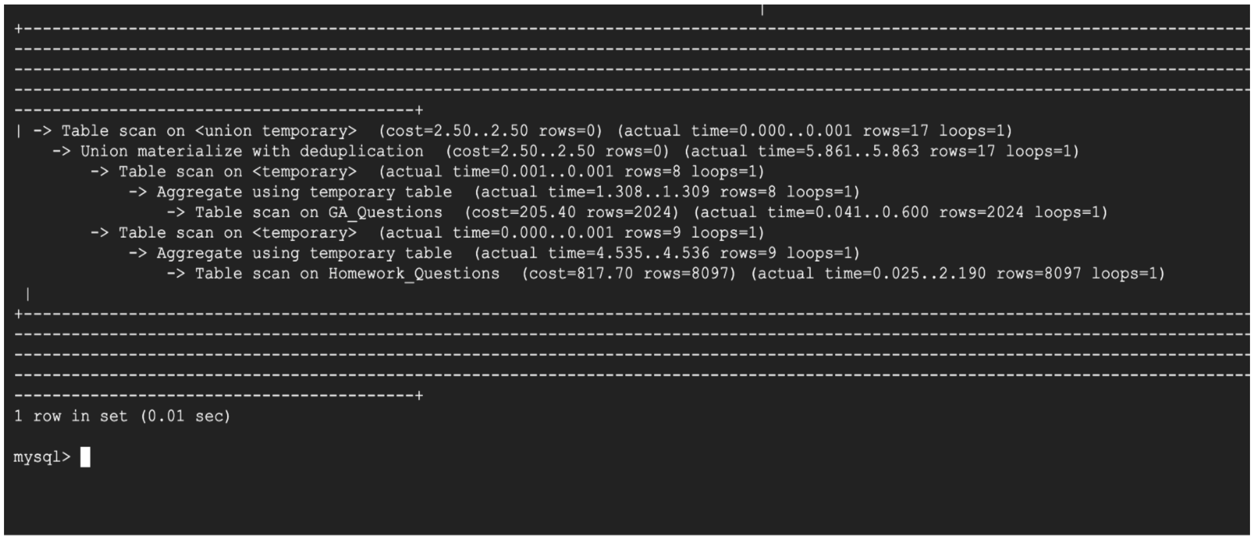

```
+-------+-------+-----------------+---------------------+
| ga_id | hw_id | question_number | AVG(question_score) |
+-------+-------+-----------------+---------------------+
|     0 |  NULL |               1 |  15.351778656126482 |
|     1 |  NULL |               1 |   16.85375494071146 |
|     0 |  NULL |               2 |  15.114624505928854 |
|     1 |  NULL |               2 |   16.58102766798419 |
|     0 |  NULL |               3 |  15.699604743083004 |
|     1 |  NULL |               3 |  15.683794466403162 |
|     0 |  NULL |               4 |   15.58498023715415 |
|     1 |  NULL |               4 |   15.66403162055336 |
|  NULL |     0 |               0 |                   0 |
|  NULL |     0 |               1 |  17.374505928853754 |
|  NULL |     1 |               1 |   17.14624505928854 |
|  NULL |     0 |               2 |  17.236166007905137 |
|  NULL |     1 |               2 |   17.33794466403162 |
|  NULL |     0 |               3 |  17.037549407114625 |
|  NULL |     1 |               3 |   16.67193675889328 |
|  NULL |     0 |               4 |   17.09090909090909 |
|  NULL |     1 |               4 |  17.221343873517785 |
+-------+-------+-----------------+---------------------+
17 rows in set (0.00 sec)
```

Result from the second SQL Query


```
+-----------+--------------+
| uin       | avg_ga_score |
+-----------+--------------+
| 600000988 |      96.0000 |
| 600000518 |      96.0000 |
| 600000013 |      96.0000 |
| 600000825 |      96.0000 |
| 600000421 |      91.0000 |
| 600000155 |      91.0000 |
| 600000076 |      91.0000 |
| 600000175 |      91.0000 |
| 600000785 |      90.0000 |
| 600000339 |      90.0000 |
| 600000862 |      90.0000 |
| 600000578 |      90.0000 |
| 600000506 |      90.0000 |
| 600000460 |      90.0000 |
| 600000716 |      90.0000 |
| 600000725 |      90.0000 |
| 600000120 |      89.0000 |
```

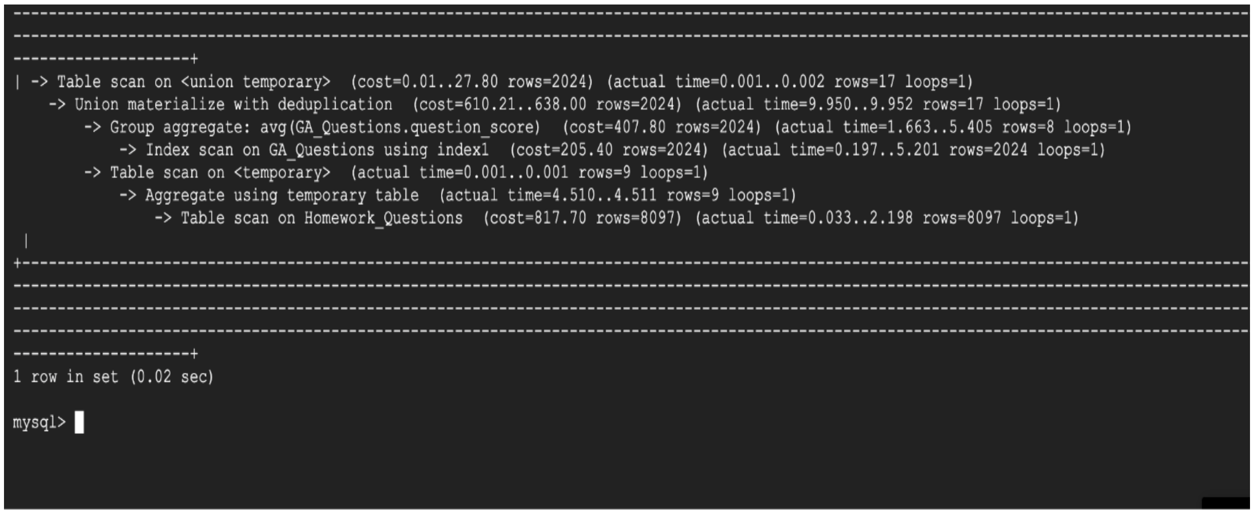
4. INDEXING:

(First Query) EXPLAIN ANALYZE WITHOUT INDEXING:



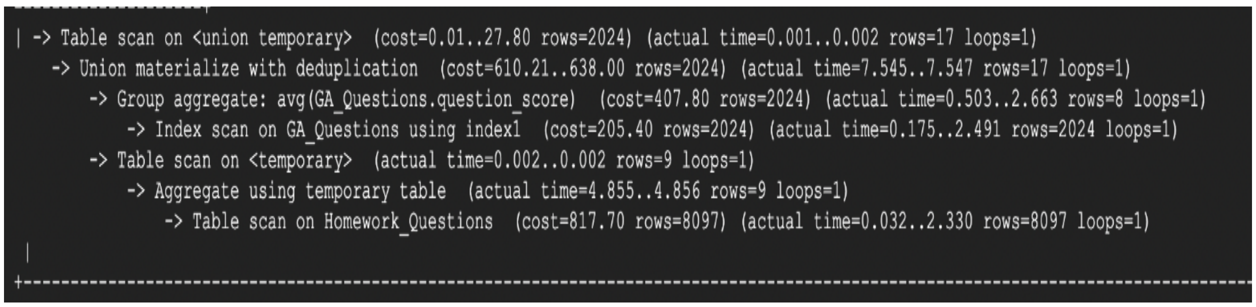
```
+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------+
| -> Table scan on <union temporary>  (cost=2.50..2.50 rows=0) (actual time=0.000..0.001 rows=17 loops=1)
    -> Union materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=5.861..5.863 rows=17 loops=1)
        -> Table scan on <temporary>  (actual time=0.001..0.001 rows=8 loops=1)
            -> Aggregate using temporary table  (actual time=1.308..1.309 rows=8 loops=1)
                -> Table scan on GA_Questions  (cost=205.40 rows=2024) (actual time=0.041..0.600 rows=2024 loops=1)
        -> Table scan on <temporary>  (actual time=0.000..0.001 rows=9 loops=1)
            -> Aggregate using temporary table  (actual time=4.535..4.536 rows=9 loops=1)
                -> Table scan on Homework_Questions  (cost=817.70 rows=8097) (actual time=0.025..2.190 rows=8097 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------+
1 row in set (0.01 sec)

mysql> 
```

(First Query) INDEX 1: WE INDEX ON GA_Questions(ga_id)



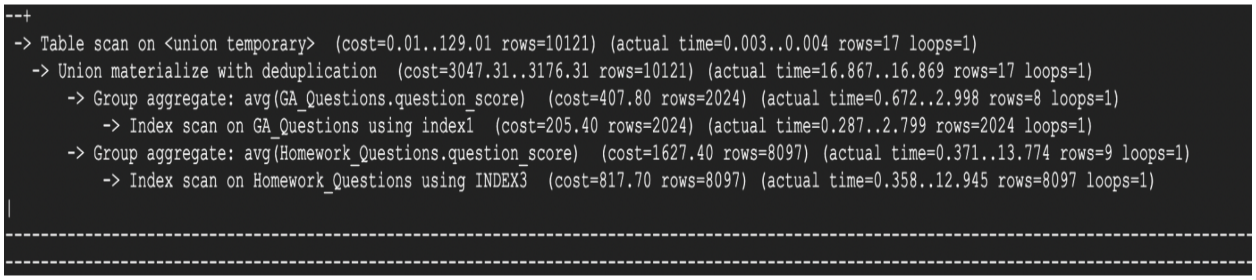
```
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
--------------------+
| -> Table scan on <union temporary>  (cost=0.01..27.80 rows=2024) (actual time=0.001..0.002 rows=17 loops=1)
    -> Union materialize with deduplication  (cost=610.21..638.00 rows=2024) (actual time=9.950..9.952 rows=17 loops=1)
        -> Group aggregate: avg(GA_Questions.question_score)  (cost=407.80 rows=2024) (actual time=1.663..5.405 rows=8 loops=1)
            -> Index scan on GA_Questions using index1  (cost=205.40 rows=2024) (actual time=0.197..5.201 rows=2024 loops=1)
        -> Table scan on <temporary>  (actual time=0.001..0.001 rows=9 loops=1)
            -> Aggregate using temporary table  (actual time=4.510..4.511 rows=9 loops=1)
                -> Table scan on Homework_Questions  (cost=817.70 rows=8097) (actual time=0.033..2.198 rows=8097 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
--------------------+
1 row in set (0.02 sec)

mysql> 
```

(First Query) INDEX 2: WE INDEX ON GA_Questions(question_number)



```
----------------------+
| -> Table scan on <union temporary>  (cost=0.01..27.80 rows=2024) (actual time=0.001..0.002 rows=17 loops=1)
    -> Union materialize with deduplication  (cost=610.21..638.00 rows=2024) (actual time=7.545..7.547 rows=17 loops=1)
        -> Group aggregate: avg(GA_Questions.question_score)  (cost=407.80 rows=2024) (actual time=0.503..2.663 rows=8 loops=1)
            -> Index scan on GA_Questions using index1  (cost=205.40 rows=2024) (actual time=0.175..2.491 rows=2024 loops=1)
        -> Table scan on <temporary>  (actual time=0.002..0.002 rows=9 loops=1)
            -> Aggregate using temporary table  (actual time=4.855..4.856 rows=9 loops=1)
                -> Table scan on Homework_Questions  (cost=817.70 rows=8097) (actual time=0.032..2.330 rows=8097 loops=1)
 |
+----------------------------------------------------------------------------------------------------------------------------
```

(First Query) INDEX 3: WE INDEX ON Homework_Questions(hw_id)



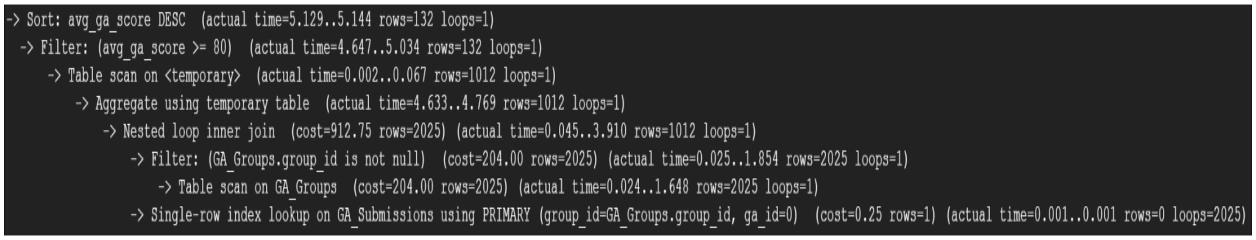
```
--+
 -> Table scan on <union temporary>  (cost=0.01..129.01 rows=10121) (actual time=0.003..0.004 rows=17 loops=1)
   -> Union materialize with deduplication  (cost=3047.31..3176.31 rows=10121) (actual time=16.867..16.869 rows=17 loops=1)
       -> Group aggregate: avg(GA_Questions.question_score)  (cost=407.80 rows=2024) (actual time=0.672..2.998 rows=8 loops=1)
           -> Index scan on GA_Questions using index1  (cost=205.40 rows=2024) (actual time=0.287..2.799 rows=2024 loops=1)
       -> Group aggregate: avg(Homework_Questions.question_score)  (cost=1627.40 rows=8097) (actual time=0.371..13.774 rows=9 loops=1)
           -> Index scan on Homework_Questions using INDEX3  (cost=817.70 rows=8097) (actual time=0.358..12.945 rows=8097 loops=1)
|
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
```

Explanation on the first SQL query:

The cost for table scan on the UNION goes down and the aggregate using a temporary table goes down.

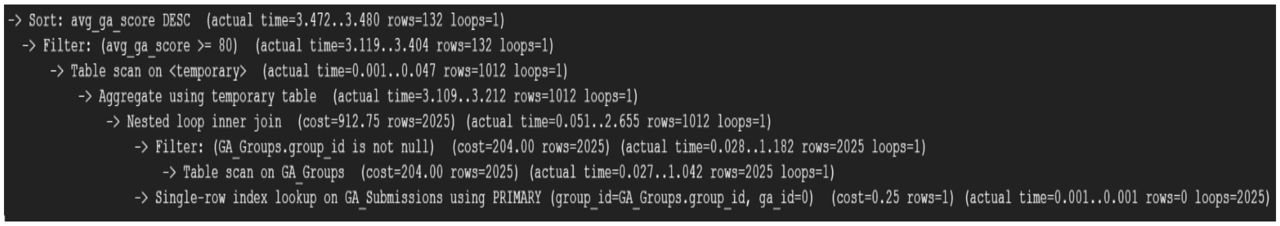
We speculate that the cost of scanning goes down because of the newly created index ga_id. In the query we were using, there is **`GROUP BY ga_id, question_number`** and this may have something to do with the mechanism of SQL doing **`GROUP BY`**. Since group by is looking for the same values inside one field, the aggregation relies on the hash table structure to look for the rows with the same values. If the “group by” is done without an index, it is expected that the SQL command will iterate over the whole table each time and the complexity of grouping will be $O(n)$. And considering the number of different values in the field we are aggregating is also $O(n)$, the overall complexity of grouping would be $O(n^2)$. However, if the hash table structure is used in one of the field to be aggregated, the complexity of aggregation would drop to $O(n)$. We can see that ga_id, question_number and hw_id are on the equivalent position in terms of aggregating. So it makes sense that the above three indexing techniques ended up reducing the running time to the same extent.

<u>(Second Query) EXPLAIN ANALYZE WITHOUT INDEXING:-</u>



```
-> Sort: avg_ga_score DESC  (actual time=5.129..5.144 rows=132 loops=1)
  -> Filter: (avg_ga_score >= 80)  (actual time=4.647..5.034 rows=132 loops=1)
      -> Table scan on <temporary>  (actual time=0.002..0.067 rows=1012 loops=1)
          -> Aggregate using temporary table  (actual time=4.633..4.769 rows=1012 loops=1)
              -> Nested loop inner join  (cost=912.75 rows=2025) (actual time=0.045..3.910 rows=1012 loops=1)
                  -> Filter: (GA_Groups.group_id is not null)  (cost=204.00 rows=2025) (actual time=0.025..1.854 rows=2025 loops=1)
                      -> Table scan on GA_Groups  (cost=204.00 rows=2025) (actual time=0.024..1.648 rows=2025 loops=1)
                  -> Single-row index lookup on GA_Submissions using PRIMARY (group_id=GA_Groups.group_id, ga_id=0)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=0 loops=2025)
```

<u>(Second Query) CREATE INDEX INDEX4 ON GA_Submissions(group_id);</u>

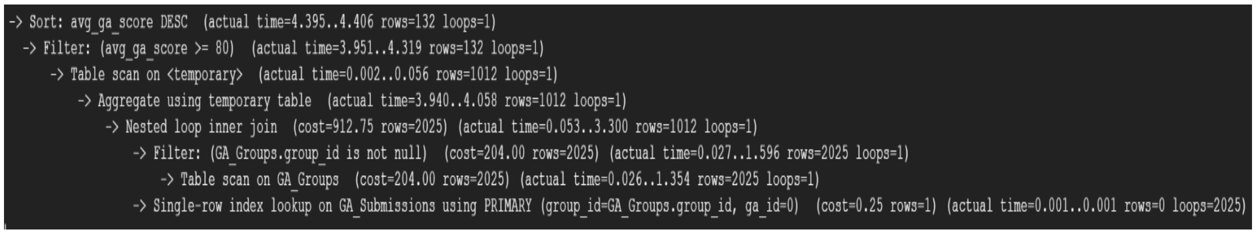


```
-> Sort: avg_ga_score DESC  (actual time=3.472..3.480 rows=132 loops=1)
  -> Filter: (avg_ga_score >= 80)  (actual time=3.119..3.404 rows=132 loops=1)
      -> Table scan on <temporary>  (actual time=0.001..0.047 rows=1012 loops=1)
          -> Aggregate using temporary table  (actual time=3.109..3.212 rows=1012 loops=1)
              -> Nested loop inner join  (cost=912.75 rows=2025) (actual time=0.051..2.655 rows=1012 loops=1)
                  -> Filter: (GA_Groups.group_id is not null)  (cost=204.00 rows=2025) (actual time=0.028..1.182 rows=2025 loops=1)
                      -> Table scan on GA_Groups  (cost=204.00 rows=2025) (actual time=0.027..1.042 rows=2025 loops=1)
                  -> Single-row index lookup on GA_Submissions using PRIMARY (group_id=GA_Groups.group_id, ga_id=0)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=0 loops=2025)
```

As you can see, the actual time went down from 4.647 to 3.119 which is a drastic improvement. This is because we natural join GA_groups and GA_Submissions on group_id. When we Natural Join, we are checking for equality of GA_groups.group_id = GA_Submissions_group_id. Indexing on group_id therefore makes it easier to lookup the value faster using the internal B-tree structure.

We drop Index4 to check the individual effect of indexing on Index5.

<u>(Second Query) CREATE INDEX INDEX5 ON GA_Submissions(score);</u>

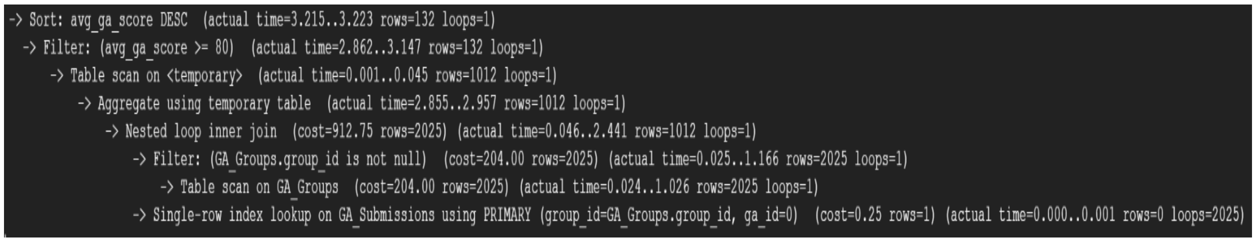


```
-> Sort: avg_ga_score DESC  (actual time=4.395..4.406 rows=132 loops=1)
  -> Filter: (avg_ga_score >= 80)  (actual time=3.951..4.319 rows=132 loops=1)
      -> Table scan on <temporary>  (actual time=0.002..0.056 rows=1012 loops=1)
          -> Aggregate using temporary table  (actual time=3.940..4.058 rows=1012 loops=1)
              -> Nested loop inner join  (cost=912.75 rows=2025) (actual time=0.053..3.300 rows=1012 loops=1)
                  -> Filter: (GA_Groups.group_id is not null)  (cost=204.00 rows=2025) (actual time=0.027..1.596 rows=2025 loops=1)
                      -> Table scan on GA_Groups  (cost=204.00 rows=2025) (actual time=0.026..1.354 rows=2025 loops=1)
                  -> Single-row index lookup on GA_Submissions using PRIMARY (group_id=GA_Groups.group_id, ga_id=0)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=0 loops=2025)
```

The actual time went down from 4.647 to 3.951 which is an improvement but not as drastic as the other indexes. This improvement could be possible because we need to search for the ga_scores for each uin and find the average score. Indexing on something we search for improves access times due to the internal nature of B-tree as explained. In the future, it may be better to index on average score itself and keep track of that data in a table since that is the attribute in our where clause that needs to be greater than or equal to 80. Having a B-Tree structure for this could help us find values greater than 8- much faster than having to linearly search the database for that score value.

We drop Index5 to check the individual effect of indexing on Index6.

<u>(Second Query) CREATE INDEX INDEX6 ON GA_Group(uin);</u>



```
-> Sort: avg_ga_score DESC  (actual time=3.215..3.223 rows=132 loops=1)
  -> Filter: (avg_ga_score >= 80)  (actual time=2.862..3.147 rows=132 loops=1)
      -> Table scan on <temporary>  (actual time=0.001..0.045 rows=1012 loops=1)
          -> Aggregate using temporary table  (actual time=2.855..2.957 rows=1012 loops=1)
              -> Nested loop inner join  (cost=912.75 rows=2025) (actual time=0.046..2.441 rows=1012 loops=1)
                  -> Filter: (GA_Groups.group_id is not null)  (cost=204.00 rows=2025) (actual time=0.025..1.166 rows=2025 loops=1)
                      -> Table scan on GA_Groups  (cost=204.00 rows=2025) (actual time=0.024..1.026 rows=2025 loops=1)
                  -> Single-row index lookup on GA_Submissions using PRIMARY (group_id=GA_Groups.group_id, ga_id=0)  (cost=0.25 rows=1) (actual time=0.000..0.001 rows=0 loops=2025)
```

The time for the outermost filter went down from 4.647 to 2.862 seconds. When we index on the GROUP BY uin column, we are constantly checking the equality of the uin values against each other. This causes a great reduction in the time taken to repeatedly find the element within the B-tree.