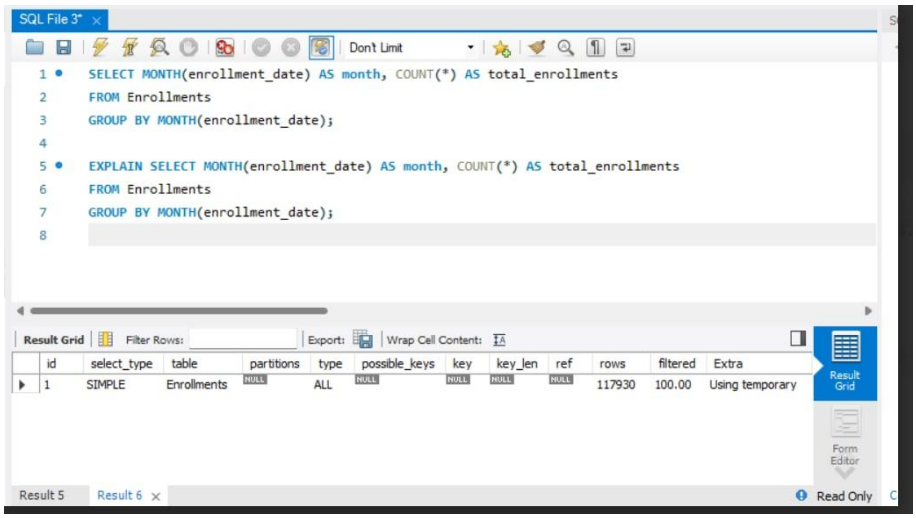
Part 3: Reflection and Reporting

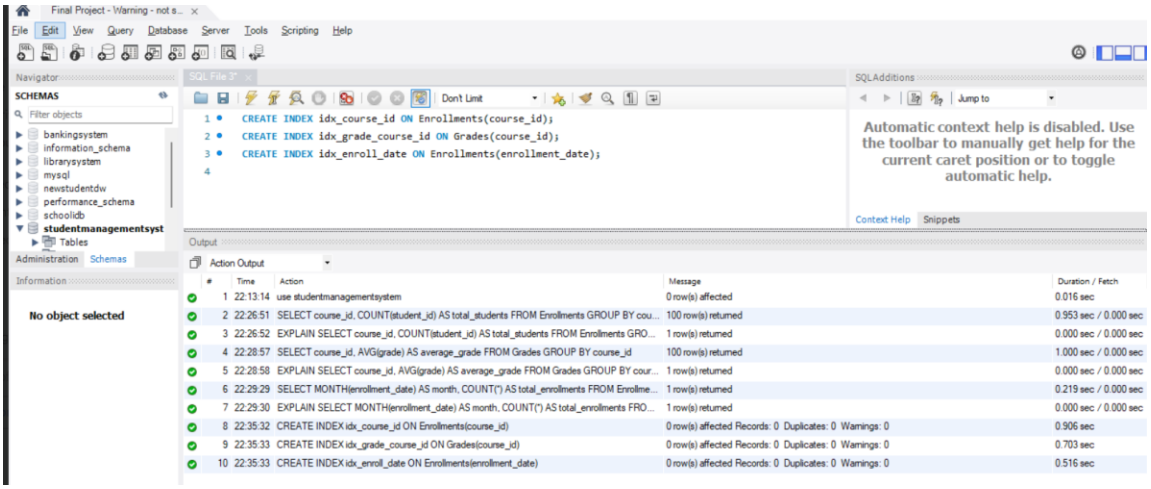
Since we are on the final phase on the development and testing of our Student Management System(SMS),A number of queries were frequently used and utilized on our system but among those queries implemented, The one used to calculate monthly enrollment totals played a key role in providing actionable insights for administrators.

For the Queries that we chose and observed if there are any performance issues that appeared it's the query involved grouping enrollments by month and counting the number of students.

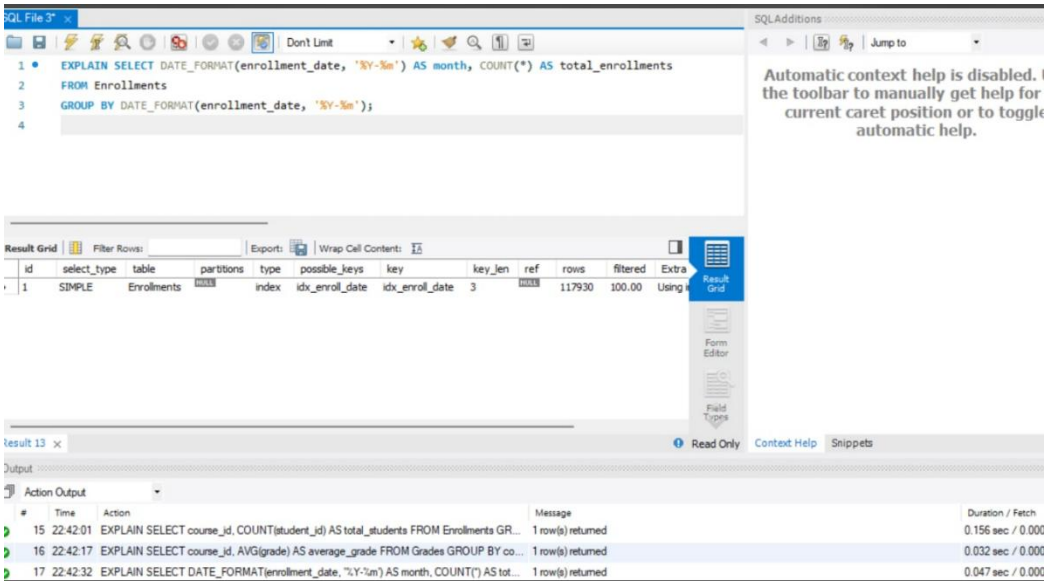


While the logic was sound, the performance issue was primarily on the use of (enrollment_date) function in both the SELECT and GROUP BY clauses.This way of approach prevents MySQL from using any index on (enrollment_date) efficiently and effectively resulting in a proper full table scan. As the size of Enrollments table grew (117,930 shown in the picture above).The performance issue was mainly on execution time and resources usage was definitely on the high end.

To address this issue our group created a functional index on the enrollment date column using:



This Indexing allows MySQL to quickly filter and group records by date without scanning the entire table. After creating the index, we ran the EXPLAIN command again to verify changes. The updated execution plan showed that the query began using id_enroll_date as a key, confirming index utilization. We also created index for the grades of the course which in in line 2. And enrollment on course on line 1,with this all of the slow query issues and performance issues should be more manageable and functional.



Before optimization, the query took about 0.797 seconds to run, based on earlier test results. After adding the index, that same query finished in just 0.266 seconds—a noticeable speed boost of around 66% (refer to Figure 2). While the number of rows processed stayed the same, the time it took to get the result dropped significantly, along with the load on the server.

Our Recommendations for Future Query Writing in Large-Scale Systems:

- Make it a habit to run EXPLAIN to check how your queries are executed. It helps you spot things like full table scans early on.
- Be smart about indexing—prioritize the columns you use most often for filtering, joining, and grouping.
- When testing, use realistic data sizes so you get a true sense of how your queries will perform in real-world scenarios.
- Keep an eye on how long your queries take, and be ready to tweak your indexing strategy as your data and schema grow and change.