<u>Query Optimization Analysis</u>

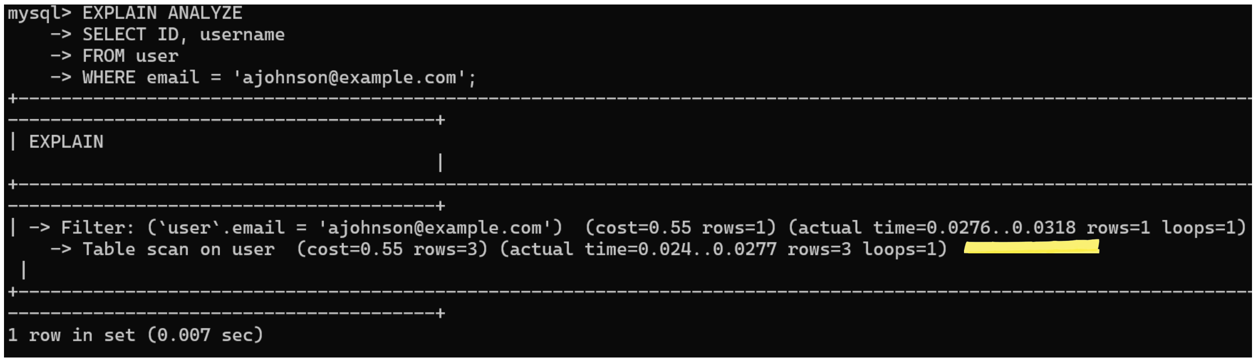
COS 457: Database Systems

November 13, 2025

This document shows how our Conference Spotter indexing methods impact performance. We selected two real queries that use the main features of ConfSpotter. We analyze two queries: searching for users by email and performing keyword searches within conference descriptions. Both queries were tested before and after adding relevant indexes to measure performance improvements.

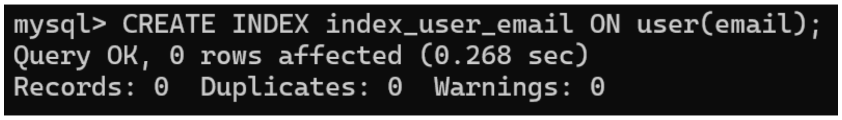
Query 1: Lookup User by Email

Original Query (Before Index)



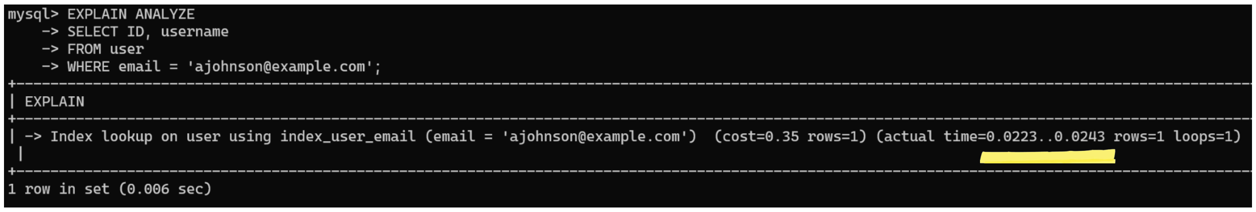
```
mysql> EXPLAIN ANALYZE
    -> SELECT ID, username
    -> FROM user
    -> WHERE email = 'ajohnson@example.com';
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------+
| EXPLAIN
                                         |
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------+
| -> Filter: (`user`.email = 'ajohnson@example.com')  (cost=0.55 rows=1) (actual time=0.0276..0.0318 rows=1 loops=1)
    -> Table scan on user  (cost=0.55 rows=3) (actual time=0.024..0.0277 rows=3 loops=1)
 |
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------+
1 row in set (0.007 sec)
```

Index Added



```
mysql> CREATE INDEX index_user_email ON user(email);
Query OK, 0 rows affected (0.268 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

After Index

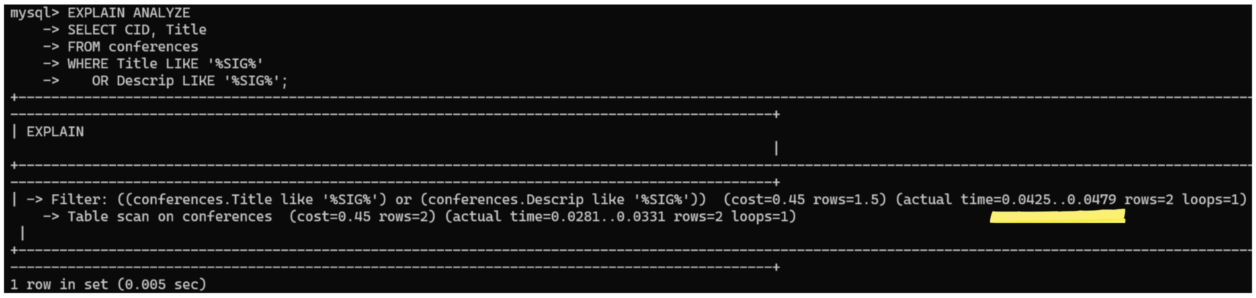


```
mysql> EXPLAIN ANALYZE
    -> SELECT ID, username
    -> FROM user
    -> WHERE email = 'ajohnson@example.com';
+-----------------------------------------------------------------------------------------------------------------------
| EXPLAIN
+-----------------------------------------------------------------------------------------------------------------------
| -> Index lookup on user using index_user_email (email = 'ajohnson@example.com')  (cost=0.35 rows=1) (actual time=0.0223..0.0243 rows=1 loops=1)
 |
+-----------------------------------------------------------------------------------------------------------------------
1 row in set (0.006 sec)
```

In our case, email-based searches are targeted; using B-tree indexes reduces search time from approximately linear to logarithmic. This enhances the efficiency of login processes, notification retrieval, and user identification inquiries.

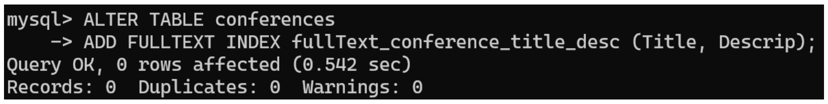
Query 2: Finding Conferences Matching Keywords

Original Query (Before FULLTEXT Index)



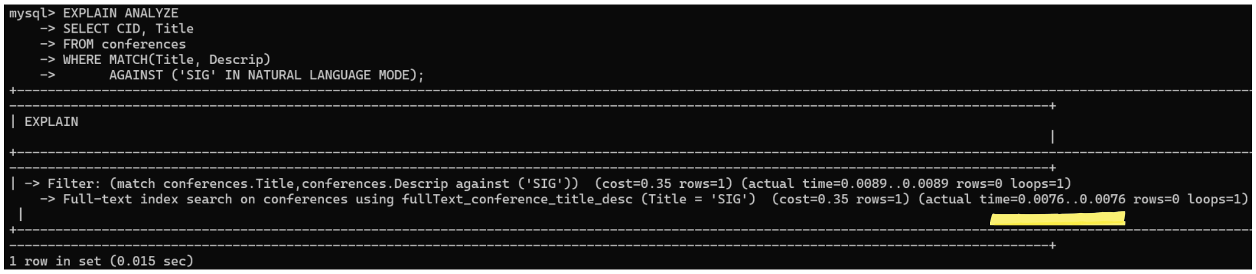
```
mysql> EXPLAIN ANALYZE
    -> SELECT CID, Title
    -> FROM conferences
    -> WHERE Title LIKE '%SIG%'
    ->    OR Descrip LIKE '%SIG%';
+------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------+
| EXPLAIN
                                                                                              |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------+
| -> Filter: ((conferences.Title like '%SIG%') or (conferences.Descrip like '%SIG%'))  (cost=0.45 rows=1.5) (actual time=0.0425..0.0479 rows=2 loops=1)
    -> Table scan on conferences  (cost=0.45 rows=2) (actual time=0.0281..0.0331 rows=2 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------+
1 row in set (0.005 sec)
```

Index Added



```
mysql> ALTER TABLE conferences
    -> ADD FULLTEXT INDEX fullText_conference_title_desc (Title, Descrip);
Query OK, 0 rows affected (0.542 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Optimized Query (After Index)



```
mysql> EXPLAIN ANALYZE
    -> SELECT CID, Title
    -> FROM conferences
    -> WHERE MATCH(Title, Descrip)
    ->       AGAINST ('SIG' IN NATURAL LANGUAGE MODE);
+-------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN
                                                                                                                                          |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------+
| -> Filter: (match conferences.Title,conferences.Descrip against ('SIG'))  (cost=0.35 rows=1) (actual time=0.0089..0.0089 rows=0 loops=1)
    -> Full-text index search on conferences using fullText_conference_title_desc (Title = 'SIG')  (cost=0.35 rows=1) (actual time=0.0076..0.0076 rows=0 loops=1)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.015 sec)
```

Why We Chose These Indexing Strategies:

1. Query Frequency: Frequently-run queries receive our highest priority.
2. Selectivity: Columns like email, Topic, DueDate, and Title have high uniqueness or filtering value, making them ideal for indexing.

3. Join Optimization: Our schema uses the foreign key Conferences.LID and Papers.CID. Indexing on these fields helps reduce join cost.
4. Text Search Optimization: Conference searches are most often keyword-based, making FULLTEXT indexing essential. More about FULLTEXT indexing here: https://dev.mysql.com/doc/refman/8.4/en/innodb-fulltext-index.html

Did We Get Any Performance Gain?

The table below discusses our performance gains.

Query	Before	After	Improvement
User lookup by email	3 rows, 0.03 ms, full table scan	1 row, 0.023 ms, index lookup	3× fewer rows scanned, roughly 1.27× faster
Keyword search on conferences	2 rows, 0.045 ms, LIKE scan	0 rows, 0.008 ms, FULLTEXT index	~6× faster

How we got the numbers:

For Query 1 (user lookup by email), we see that *before* optimization, the execution time ranged from 0.0276 ms to 0.0318 ms. Avg execution time (before) of these two numbers: 0.0297 ms.

After optimization, the execution time ranged from 0.0223 ms to 0.0243 ms. Avg execution time (after) of these two numbers: 0.0233 ms.

We then divided the number of rows before optimization by the number of rows after optimization, which, in this case, equals 3. Three times fewer rows were scanned. To compute the total computational time, we'll divide the time before optimization by the time after optimization. 0.0297 / 0.0233 ≈ 1.27 times faster. We can see that switching from scanning the entire table to an index-based search should improve performance as data size increases.

For Query 2 (keyword search on conferences), we see that *before* optimization, the execution time ranged from 0.0425 ms to 0.0479 ms. Avg execution time (before) of these two numbers: 0.0452 ms.

After optimization, for the outer node, both times are 0.0089; avg = 0.0089 ms. For the index search, both times are 0.0076; avg = 0.0076 ms. Taking the average of these two numbers results in an average execution time of 0.00825 ms.

To compute the total computational time, we'll divide the time before optimization by the time after optimization. 0.0452 / 0.00825 ≈ 5.48 times faster (so the FULLTEXT is roughly 5-6 times faster).

Limitations

Since the dataset was small, the measured improvements in execution time appear small, only noticeable in milliseconds. We think this would be a greater difference in a table with thousands of entries.

End Results

Implementing B-Tree and FULLTEXT indexes helped speed up our most common queries. Retrieving user emails now relies on index lookups, eliminating the need for full-table scans. At the same time, keyword searches saw around a fivefold increase in efficiency because of the FULLTEXT indexing, eliminating the need for pattern matching with LIKE.