

Gage White

## Index and Query Optimization Analysis

Below are performance results for two queries of the Lobster Notes database. Unfortunately, due to difficulties importing unique scraped data into the database, using the most effective queries and indexing strategies was limited. This resulted in the attributes with the widest range of values to be the ‘Date’ and ‘DateFor’ attributes under ‘Resource.’ In addition, the ‘Resource’ table was populated with relatively few entries, just under 1700. This resulted in fast queries no matter what.

Pictured below is the first query, seeking a range in ‘DateFor’, the attribute describing what lecture date the resource is from. The ‘DateFor’ dates were randomized in each entry, so some were in the future.

```
EXPLAIN
SELECT
    ResourceID,
    Topic,
    DateFor,
    Author
FROM Resource
WHERE DateFor >= '2025-10-01' AND DateFor <= '2026-11-30'
ORDER BY DateFor DESC;
```

### Timing Before Index:

**Timing (as measured at client side):**  
Execution time: 0:00:0.01600000

**Timing (as measured by the server):**  
Execution time: 0:00:0.01258950  
Table lock wait time: 0:00:0.00000400

Although the timing change is not significant, there is enough of a change between the two measurements that I am confident the index will be effective with a larger data set.

### Timing After Index:

**Timing (as measured at client side):**  
Execution time: 0:00:0.00000000

**Timing (as measured by the server):**  
Execution time: 0:00:0.00041210  
Table lock wait time: 0:00:0.00000400

Pictured below is the second query. Since the only other non-date attribute with a wide range of possible values was Author, which is indexed automatically, I queried ‘Date’, the attribute that describes the date the resource was posted. Like the ‘DateFor’, I randomized the values to provide a wider range of values to search.

- [EXPLAIN](#)

```
SELECT
    ResourceID,
    Topic,
    Date
FROM Resource
WHERE Date >= '2025-10-01' AND Date <= '2026-11-30'
ORDER BY Date DESC;
```

### Timing Before Indexing:

#### Query Statistics

##### Timing (as measured at client side):

Execution time: 0:00:0.00000000

##### Timing (as measured by the server):

Execution time: 0:00:0.00037590

Table lock wait time: 0:00:0.00000300

### Timing After Indexing:

##### Timing (as measured at client side):

Execution time: 0:00:0.00000000

##### Timing (as measured by the server):

Execution time: 0:00:0.00043610

Table lock wait time: 0:00:0.00000300

Unfortunately, the timing results were even worse after the indexing, however, I expect once the database is more populated, the index will prove to be more useful.

I expect once the database is more populated, the indexes created for catalog number, topic, keywords and so on will prove to be more useful.