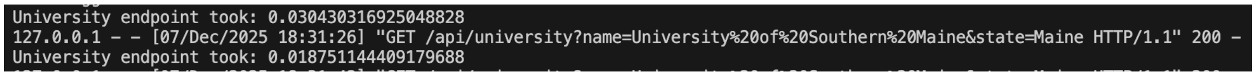
Campus Insider Performance Analysis & Optimization Report
COS 457/ COS 557
Professor Behrooz Mansouri

<u>Bottleneck 1 - Nathaniel</u>

Created indices between rooms/LID and buildings/LID



```
University endpoint took: 0.030430316925048828
127.0.0.1 - - [07/Dec/2025 18:31:26] "GET /api/university?name=University%20of%20Southern%20Maine&state=Maine HTTP/1.1" 200 -
University endpoint took: 0.018751144409179688
```

This increased the performance of queries that utilize these joins by a little under 40%

During load testing and endpoint tracing, we discovered that the university locations endpoint (used for searching buildings and rooms belonging to a university) utilized some join-heavy queries that could potentially affect performance. This potential bottleneck was a result of the following joins:

```
JOIN rooms R        ON L.LID = R.LID
JOIN buildings B    ON L.LID = B.LID
```

- rooms(LID) had no index
- buildings(LID) had no index

This forced MySQL to perform full table scans on both tables whenever the endpoint was called, even when requesting a single room. Our current dataset is small, but as it grows (thousands of rooms across ~3000+ US colleges), this would scale very poorly.

<u>Optimization 1 - Nathaniel</u>

We added two targeted indexes:

```
CREATE INDEX idx_rooms_LID ON rooms (LID);
CREATE INDEX idx_buildings_LID ON buildings (LID)
```

These allow MySQL to perform constant-time much more efficient lookups when joining location → rooms or location → buildings

After adding the indexes, we re-ran the exact same request to the university endpoint.

<u>Before (no LID index):</u>

- University endpoint took: 0.030430316925048828 seconds

<u>After (with LID index):</u>

- University endpoint took: 0.018751144409179688 seconds

~38.3% faster

On high traffic pages where this search operation occurs constantly, this improvement yields massive savings.

Bottleneck 2 - Ben

The location search is one of the most used pages in our application due to the user needing to find the location they want to view or review. The filters they might use to find the location include campus, building, and room type along with other attributes.

Optimization 2 - Ben

We made a composite index on the university_id, campus_name, and name (of the location).

The following examples are random queries based on what a user might search for:

Result of all locations being searched without filters:

Before (no composite index):

- Endpoint search_locations took: 0.018868446350097656,

After (with composite index):

- Endpoint search_locations took: 0.007804393768310547

Result of locations being searched with room type, campus, and room size filters:

Before (no composite index):

- Endpoint search_locations took: 0.013176441192626953

After (with composite index):

- Endpoint search_locations took: 0.01009368896484375

The resulting efficiency increases are around 59% and 23% respectively.