Query Analysis for prompt:

EXPLAIN SELECT * FROM jobs

WHERE city = 'Auburn';

This is what was returned from the query before indexing:

# id, select_type, table, partitions, type, possible_keys, key, key_len, ref, rows, filtered, Extra

'1', 'SIMPLE', 'jobs', NULL, 'ALL', NULL, NULL, NULL, NULL, '3068', '10.00', 'Using where'

The ‘ALL’ section meaning that the query scans every single row in the table  before returning the data being searched. 3068 means the number of rows that are being scanned to return this query.

The following is what was returned after creating an index on city with the following SQL script:

CREATE INDEX idx_jobs_city

ON jobs(city);

EXPLAIN SELECT * FROM jobs

WHERE city = 'Auburn';

# id, select_type, table, partitions, type, possible_keys, key, key_len, ref, rows, filtered, Extra

'1', 'SIMPLE', 'jobs', NULL, 'ref', 'idx_jobs_city', 'idx_jobs_city', '403', 'const', '75', '100.00',NULL

- ‘ref’ shows that the lookup is looking through an index for city names
- 75 is the number of rows being looked through because of the new indexing. 75 is much smaller than 3068 which means significant performance gains
- Extra being ‘NULL’ is also a good thing because the query isn’t scanning any other tables, just table between indexes that it needs

The justification behind this kind of indexing – a single-column index – is that it helps MySQL quickly find all the rows that match the city we're searching for. The index keeps the city values in order, so all the same city names end up next to each other inside the index. That way, MySQL can jump straight to the group of matching cities instead of checking every row in the whole table. Also, when searching for jobs, location is a big factor in whether or not a prospective worker will want to apply or not so the city in which jobs are in will be queried a lot.

---

Query Analysis for prompt:

EXPLAIN SELECT * FROM employer

WHERE employer_name = 'LearnWell';

The following was returned by the query before indexing has been done on the employer_name column:

# id, select_type, table, partitions, type, possible_keys, key, key_len, ref, rows, filtered, Extra

'1', 'SIMPLE', 'employer', NULL, 'ALL', NULL, NULL, NULL, NULL, '552', '10.00', 'Using where'

In this information we can see that 552 rows have been scanned,  and 'ALL' means that the whole table has been scanned to return one employer name which is very inefficient.

Indexing with single-column index:

CREATE INDEX idx_employer_name

ON employer(employer_name);

EXPLAIN SELECT * FROM employer

WHERE employer_name = 'LearnWell';

# id, select_type, table, partitions, type, possible_keys, key, key_len, ref, rows, filtered, Extra

'1', 'SIMPLE', 'employer', NULL, 'ref', 'idx_employer_name', 'idx_employer_name', '803', 'const', '1', '100.00', NULL

- Now instead of ‘ALL’, in its place is ‘ref’ which means the query is only looking at the indexes it needs
- The 1 means that there is only one row that needs to be checked which is a massive improvement over 553 rows

The justification for indexing the employer_name column with a single-column index is that each employer name is unique, so a query looking for a specific employer does not need to search the entire table. Instead, MySQL can use the index to find the exact row directly, as we saw above where only one row is searched to return the employer name. Indexing this column is important because in the real world it would be queried very often. For example, when employers post many jobs under one company and prospective employees want to search for jobs by employer or within a particular industry.