Query 1

This query returns the name of the companies who own a license where a transmit location is in Chicago, or a receiver location is in New York.

```
7  •  SELECT DISTINCT License.name
8     FROM License NATURAL JOIN Path JOIN Locations ON Locations.location_number= Path.transmit_location_number
9     WHERE Locations.location_city = 'Chicago'
10    UNION
11    SELECT DISTINCT License.name
12    FROM License NATURAL JOIN Path JOIN Locations ON Locations.location_number= Path.receiver_location_number
13    WHERE Locations.location_city = 'New York'
14    LIMIT 15;
15
16
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name
THE ISLAND TELEPHONE COMPANY
UNION PACIFIC RAILROAD
NORFOLK SOUTHERN RAILWAY COMPANY
SOUTHERN NEW ENGLAND TELEPHONE COMPANY
GRAY TELEVISION LICENSEE, LLC
TEGNA East Coast Broadcasting, LLC
Massachusetts Commonwealth of, Department ...
ILLINOIS CENTRAL RAILROAD COMPANY
COMMUNICATIONS COUNSEL GROUP, INC.
CALIFORNIA, STATE OF
MASSACHUSETTS, COMMONWEALTH OF
NSTAR Electric Company
Eversource Energy Service Company
NORTHEAST UTILITIES SERVICE COMPANY
PACIFIC AND SOUTHERN, LLC

Result Grid | Form Editor | Field Types | Query Stats

No Additional Index

```
1  -> Table scan on <union temporary>  (cost=0.01..19.91 rows=1394) (actual time=0.001..0.041 rows=639 loops=1)
2     -> Union materialize with deduplication  (cost=21227.03..21246.93 rows=1394) (actual time=26.099..26.174 rows=639 loops=1)
3        -> Table scan on <temporary>  (cost=0.02..11.20 rows=697) (actual time=0.001..0.044 rows=639 loops=1)
4           -> Temporary table with deduplication  (cost=10532.65..10543.83 rows=697) (actual time=12.357..12.442 rows=639 loops=1)
5              -> Inner hash join (Locations.location_number = `Path`.transmit_location_number)  (cost=10462.96 rows=697) (actual time=4.235..6.645 rows=14185 loops=1)
6                 -> Filter: (Locations.location_city = 'Chicago')  (cost=0.78 rows=25) (actual time=0.070..0.912 rows=16 loops=1)
7                    -> Table scan on Locations  (cost=0.78 rows=2456) (actual time=0.058..0.864 rows=2456 loops=1)
8                 -> Hash
9                    -> Nested loop inner join  (cost=1287.83 rows=2837) (actual time=0.049..3.614 rows=2837 loops=1)
10                      -> Table scan on Path  (cost=294.88 rows=2837) (actual time=0.038..1.080 rows=2837 loops=1)
11                      -> Single-row index lookup on License using PRIMARY (unique_system_identifier=`Path`.unique_system_identifier)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=2837)
12    -> Table scan on <temporary>  (cost=0.02..11.20 rows=697) (actual time=0.002..0.044 rows=636 loops=1)
13       -> Temporary table with deduplication  (cost=10532.65..10543.83 rows=697) (actual time=13.095..13.172 rows=636 loops=1)
14          -> Inner hash join (Locations.location_number = `Path`.receiver_location_number)  (cost=10462.96 rows=697) (actual time=3.221..5.805 rows=19528 loops=1)
15             -> Filter: (Locations.location_city = 'New York')  (cost=0.78 rows=25) (actual time=0.039..0.791 rows=21 loops=1)
16                -> Table scan on Locations  (cost=0.78 rows=2456) (actual time=0.037..0.588 rows=2456 loops=1)
17             -> Hash
18                -> Nested loop inner join  (cost=1287.83 rows=2837) (actual time=0.035..2.761 rows=2837 loops=1)
19                   -> Table scan on Path  (cost=294.88 rows=2837) (actual time=0.024..0.682 rows=2837 loops=1)
20                   -> Single-row index lookup on License using PRIMARY (unique_system_identifier=`Path`.unique_system_identifier)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=2837)
21
```

Index on Locations.location_city

```
-> Table scan on <union temporary>  (cost=0.01..133.70 rows=10497) (actual time=0.001..0.041 rows=639 loops=1)
    -> Union materialize with deduplication  (cost=13953.79..14087.47 rows=10497) (actual time=32.561..32.637 rows=639 loops=1)
        -> Table scan on <temporary>  (cost=0.01..59.24 rows=4539) (actual time=0.003..0.065 rows=639 loops=1)
            -> Temporary table with deduplication  (cost=5532.46..5591.68 rows=4539) (actual time=13.099..13.204 rows=639 loops=1)
                -> Nested loop inner join  (cost=5078.53 rows=4539) (actual time=0.111..7.985 rows=14185 loops=1)
                    -> Inner hash join (`Path`.transmit_location_number = Locations.location_number)  (cost=4553.68 rows=4539) (actual time=0.095..2.325 rows=14185 loops=1)
                        -> Table scan on Path  (cost=2.49 rows=2837) (actual time=0.025..0.708 rows=2837 loops=1)
                        -> Hash
                            -> Index lookup on Locations using MYINDEX1 (location_city='Chicago')  (cost=3.02 rows=16) (actual time=0.036..0.041 rows=16 loops=1)
                    -> Single-row index lookup on License using PRIMARY (unique_system_identifier=`Path`.unique_system_identifier)  (cost=0.01 rows=1) (actual time=0.000..0.000 rows=1 loops=14185)
        -> Table scan on <temporary>  (cost=0.01..76.96 rows=5958) (actual time=0.004..0.050 rows=636 loops=1)
            -> Temporary table with deduplication  (cost=7235.45..7312.40 rows=5958) (actual time=18.677..18.763 rows=636 loops=1)
                -> Nested loop inner join  (cost=6639.67 rows=5958) (actual time=0.135..10.779 rows=19528 loops=1)
                    -> Inner hash join (`Path`.receiver_location_number = Locations.location_number)  (cost=5972.97 rows=5958) (actual time=0.125..3.166 rows=19528 loops=1)
                        -> Table scan on Path  (cost=1.90 rows=2837) (actual time=0.024..0.901 rows=2837 loops=1)
                        -> Hash
                            -> Index lookup on Locations using MYINDEX1 (location_city='New York')  (cost=3.73 rows=21) (actual time=0.061..0.065 rows=21 loops=1)
                    -> Single-row index lookup on License using PRIMARY (unique_system_identifier=`Path`.unique_system_identifier)  (cost=0.01 rows=1) (actual time=0.000..0.000 rows=1 loops=19528)

```

Index on Locations.location_city and Locations.unique_system_identifier

```
-> Table scan on <union temporary>  (cost=0.01..19.91 rows=1394) (actual time=0.001..0.044 rows=639 loops=1)
    -> Union materialize with deduplication  (cost=21227.03..21246.93 rows=1394) (actual time=25.461..25.540 rows=639 loops=1)
        -> Table scan on <temporary>  (cost=0.02..11.20 rows=697) (actual time=0.002..0.047 rows=639 loops=1)
            -> Temporary table with deduplication  (cost=10532.65..10543.83 rows=697) (actual time=11.139..11.219 rows=639 loops=1)
                -> Inner hash join (Locations.location_number = `Path`.transmit_location_number)  (cost=10462.96 rows=697) (actual time=3.620..5.850 rows=14185 loops=1)
                    -> Filter: (Locations.location_city = 'Chicago')  (cost=0.78 rows=2) (actual time=0.062..0.916 rows=16 loops=1)
                        -> Index scan on Locations using MYINDEX2  (cost=0.78 rows=2456) (actual time=0.040..0.629 rows=2456 loops=1)
                    -> Hash
                        -> Nested loop inner join  (cost=1287.83 rows=2837) (actual time=0.059..3.120 rows=2837 loops=1)
                            -> Table scan on Path  (cost=294.88 rows=2837) (actual time=0.046..0.742 rows=2837 loops=1)
                            -> Single-row index lookup on License using PRIMARY (unique_system_identifier=`Path`.unique_system_identifier)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=2837)
        -> Table scan on <temporary>  (cost=0.02..11.20 rows=697) (actual time=0.002..0.043 rows=636 loops=1)
            -> Temporary table with deduplication  (cost=10532.65..10543.83 rows=697) (actual time=13.566..13.644 rows=636 loops=1)
                -> Inner hash join (Locations.location_number = `Path`.receiver_location_number)  (cost=10462.96 rows=697) (actual time=3.581..6.251 rows=19528 loops=1)
                    -> Filter: (Locations.location_city = 'New York')  (cost=0.78 rows=2) (actual time=0.057..0.825 rows=21 loops=1)
                        -> Index scan on Locations using MYINDEX2  (cost=0.78 rows=2456) (actual time=0.049..0.581 rows=2456 loops=1)
                    -> Hash
                        -> Nested loop inner join  (cost=1287.83 rows=2837) (actual time=0.043..2.995 rows=2837 loops=1)
                            -> Table scan on Path  (cost=294.88 rows=2837) (actual time=0.028..0.720 rows=2837 loops=1)
                            -> Single-row index lookup on License using PRIMARY (unique_system_identifier=`Path`.unique_system_identifier)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=2837)

```

Index on Location.location_state

```
-> Table scan on <union temporary>  (cost=0.01..19.91 rows=1394) (actual time=0.001..0.041 rows=639 loops=1)
    -> Union materialize with deduplication  (cost=21227.03..21246.93 rows=1394) (actual time=25.250..25.325 rows=639 loops=1)
        -> Table scan on <temporary>  (cost=0.02..11.20 rows=697) (actual time=0.002..0.049 rows=639 loops=1)
            -> Temporary table with deduplication  (cost=10532.65..10543.83 rows=697) (actual time=10.867..10.952 rows=639 loops=1)
                -> Inner hash join (Locations.location_number = `Path`.transmit_location_number)  (cost=10462.96 rows=697) (actual time=3.466..5.599 rows=14185 loops=1)
                    -> Filter: (Locations.location_city = 'Chicago')  (cost=0.78 rows=2) (actual time=0.064..0.869 rows=16 loops=1)
                        -> Table scan on Locations  (cost=0.78 rows=2456) (actual time=0.048..0.630 rows=2456 loops=1)
                    -> Hash
                        -> Nested loop inner join  (cost=1287.83 rows=2837) (actual time=0.058..3.005 rows=2837 loops=1)
                            -> Table scan on Path  (cost=294.88 rows=2837) (actual time=0.042..0.710 rows=2837 loops=1)
                            -> Single-row index lookup on License using PRIMARY (unique_system_identifier=`Path`.unique_system_identifier)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=2837)
        -> Table scan on <temporary>  (cost=0.02..11.20 rows=697) (actual time=0.002..0.049 rows=636 loops=1)
            -> Temporary table with deduplication  (cost=10532.65..10543.83 rows=697) (actual time=13.665..13.747 rows=636 loops=1)
                -> Inner hash join (Locations.location_number = `Path`.receiver_location_number)  (cost=10462.96 rows=697) (actual time=3.382..6.230 rows=19528 loops=1)
                    -> Filter: (Locations.location_city = 'New York')  (cost=0.78 rows=2) (actual time=0.047..0.976 rows=21 loops=1)
                        -> Table scan on Locations  (cost=0.78 rows=2456) (actual time=0.042..0.753 rows=2456 loops=1)
                    -> Hash
                        -> Nested loop inner join  (cost=1287.83 rows=2837) (actual time=0.040..2.916 rows=2837 loops=1)
                            -> Table scan on Path  (cost=294.88 rows=2837) (actual time=0.026..0.728 rows=2837 loops=1)
                            -> Single-row index lookup on License using PRIMARY (unique_system_identifier=`Path`.unique_system_identifier)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=2837)

```

Based on the result of using Explain Analyze for the query using three different indices, it is clear that using Locations.location_city as an index was most effective. This makes sense, since in our query we are looking for specific cities. Using Locations.location_city and Locations.unique_system_identifier as single index did not increase efficiency of the query, since using Locations.unique_system_identifier as part of the index was unnecessary. Finally, using Location.location_state as the single attribute of an index didn't improve our query efficiency, since we never utilize that value in the query.

Query 2

This query returns a count of the number of communications locations that are present in each city. Only counts locations that are associated with a license where the company has an “INC.” in the name.

```
4 •  SELECT loc.location_city, COUNT(*) as Cnt
5    From License li JOIN Locations loc USING (unique_system_identifier)
6    WHERE li.name LIKE '%INC.%'
7    GROUP BY loc.location_city
8    ORDER BY Cnt DESC
9    LIMIT 15;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

location_city	Cnt
HOUSTON	18
PITTSBURGH	11
SAN FRANCISCO	10
Raleigh	10
Tampa	9
CHICAGO	9
SPOKANE	6
RIVERVIEW	6
MILWAUKEE	6
COLUMBUS	6
DENVER	6
San Antonio	6
Sacramento	6
ALBUQUERQUE	6
BIRMINGHAM	5

Result Grid | Form Editor | Field Types | Query Stats | Execution Plan

No index

```
1  -> Sort: Cnt DESC  (actual time=3.725..3.743 rows=303 loops=1)
2     -> Table scan on <temporary>  (actual time=0.001..0.024 rows=303 loops=1)
3        -> Aggregate using temporary table  (actual time=3.598..3.638 rows=303 loops=1)
4           -> Nested loop inner join  (cost=443.08 rows=483) (actual time=0.088..3.243 rows=532 loops=1)
5              -> Filter: (li.`name` like '%INC.%')  (cost=310.20 rows=339) (actual time=0.068..1.911 rows=700 loops=1)
6                 -> Table scan on li  (cost=310.20 rows=3047) (actual time=0.060..0.760 rows=3019 loops=1)
7              -> Index lookup on loc using PRIMARY (unique_system_identifier=li.unique_system_identifier)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=700)
8
```

Index on License.name

```
1  -> Sort: Cnt DESC  (actual time=3.515..3.533 rows=303 loops=1)
2     -> Table scan on <temporary>  (actual time=0.001..0.021 rows=303 loops=1)
3        -> Aggregate using temporary table  (actual time=3.400..3.438 rows=303 loops=1)
4           -> Nested loop inner join  (cost=443.08 rows=483) (actual time=0.093..3.064 rows=532 loops=1)
5              -> Filter: (li.`name` like '%INC.%')  (cost=310.20 rows=339) (actual time=0.052..1.774 rows=700 loops=1)
6                 -> Index scan on li using MYINDEX1  (cost=310.20 rows=3047) (actual time=0.044..0.639 rows=3019 loops=1)
7              -> Index lookup on loc using PRIMARY (unique_system_identifier=li.unique_system_identifier)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=700)
8
```

Index on License.name and License.unique_system_identifier

```
1  -> Sort: Cnt DESC  (actual time=3.613..3.633 rows=303 loops=1)
2      -> Table scan on <temporary>  (actual time=0.001..0.024 rows=303 loops=1)
3          -> Aggregate using temporary table  (actual time=3.494..3.534 rows=303 loops=1)
4              -> Nested loop inner join  (cost=443.08 rows=483) (actual time=0.080..3.114 rows=532 loops=1)
5                  -> Filter: (li.`name` like '%INC.%')  (cost=310.20 rows=339) (actual time=0.063..1.826 rows=700 loops=1)
6                      -> Index scan on li using MYINDEX2  (cost=310.20 rows=3047) (actual time=0.054..0.667 rows=3019 loops=1)
7                  -> Index lookup on loc using PRIMARY (unique_system_identifier=li.unique_system_identifier)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=700)
8
```

Index on License.name and License.city License.email and License.street_address

```
1  -> Sort: Cnt DESC  (actual time=3.607..3.626 rows=303 loops=1)
2      -> Table scan on <temporary>  (actual time=0.001..0.024 rows=303 loops=1)
3          -> Aggregate using temporary table  (actual time=3.482..3.523 rows=303 loops=1)
4              -> Nested loop inner join  (cost=443.08 rows=483) (actual time=0.108..3.1[illegible] rows=[illegible] loops=1)
5                  -> Filter: (li.`name` like '%INC.%')  (cost=310.20 rows=339) (actual time=0.064..1.812 rows=700 loops=1)
6                      -> Index scan on li using MYINDEX3  (cost=310.20 rows=3047) (actual time=0.053..0.669 rows=3019 loops=1)
7                  -> Index lookup on loc using PRIMARY (unique_system_identifier=li.unique_system_identifier)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=700)
8
```

For all the indices, we noticed that very little changed in terms of performance. There was a slight but noticeable improvement on the License.name index. However, I don't think the negligible improvement we saw there is related to this index. Because we use LIKE in our query with a wildcard in the beginning and end of a string, we shouldn't expect the index to help us out too much. If we were searching for an exactly specific company name, or perhaps removing the leading wildcard('INC.%'), I would expect that our performance would be better, and we see improvements similar to the first query.