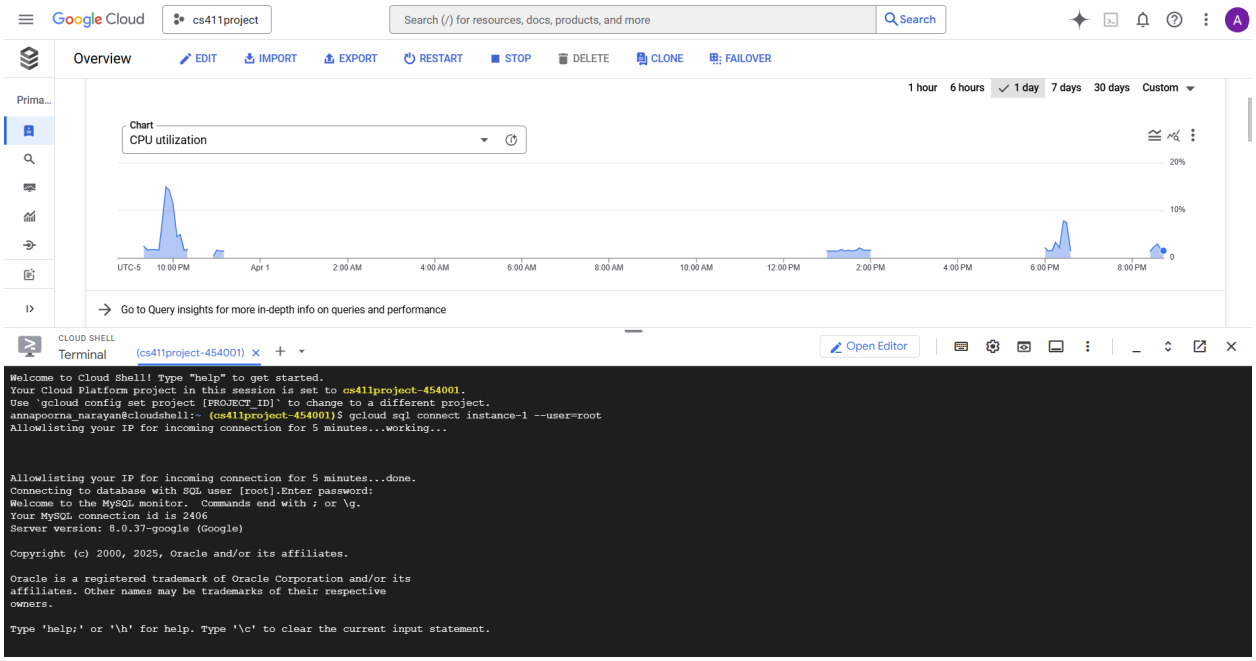
Stage 3: Database Implementation and Indexing

***GCP Connection Evidence through Shell Terminal*:**



Data Definition Language (DDL) Commands

```
Unset
SELECT DATABASE();
USE dwellx;
```

Main Tables:

```
Unset
CREATE TABLE Schools (
    school_id INT PRIMARY KEY,
    name VARCHAR(100),
```

```
    school_type VARCHAR(100),
    website VARCHAR(100),
    city VARCHAR(100),
    zip_code INT,
    address VARCHAR(250),
    phone VARCHAR(12),
    student_growth_rating VARCHAR(500),
    student_growth_description VARCHAR(500),
    student_attainment_rating VARCHAR(500),
    culture_climate_rating VARCHAR(500),
    school_latitude DECIMAL(9,6),
    school_longitude DECIMAL(9,6),
    school_survey_rating_description VARCHAR(500),
    FOREIGN KEY (zip_code) REFERENCES Housing(zip_code)
);

CREATE TABLE Housing (
    housing_id INT PRIMARY KEY,
    community_area_name VARCHAR(100),
    community_area_number INT,
    property_type VARCHAR(50),
```

```
    property_name VARCHAR(100),
    address VARCHAR(150),
    zip_code INT,
    phone_number VARCHAR(12),
    management_company VARCHAR(50),
    units INT,
    x_coord DECIMAL(9,3),
    y_coord DECIMAL(9,3),
    latitude DECIMAL(9,6),
    longitude DECIMAL(9,6)
);

CREATE TABLE Hospitals (
    hospital_id INT PRIMARY KEY,
    name VARCHAR(225),
    facility VARCHAR(225),
    address VARCHAR(225),
    type VARCHAR(225),
    parent_org VARCHAR(225),
    zip_code INT,
    ward INT,
```

```
    tract INT,
    area_number INT,
    community INT,
    FOREIGN KEY (zip_code) REFERENCES Housing(zip_code)
);

CREATE TABLE Crimes (
    case_number INT PRIMARY KEY,
    date_of_occurrence DATE,
    zip_code INT,
    iucr VARCHAR(225),
    latitude DECIMAL(9,6),
    longitude DECIMAL(9,6),
    arrest VARCHAR(500),
    FOREIGN KEY (zip_code) REFERENCES Housing(zip_code)
);

CREATE TABLE Parks (
    object_id INT PRIMARY KEY,
    zip_code INT,
    park VARCHAR(500),
```

```
    facility VARCHAR(500),
    facility_type VARCHAR(500),
    x_coord DECIMAL(8,6),
    y_coord DECIMAL(9,6),
    FOREIGN KEY (zip_code) REFERENCES Housing(zip_code)
);
CREATE TABLE users (
        user_id VARCHAR(500) PRIMARY KEY,
        email VARCHAR(225),
        display_name VARCHAR(225),
        profile_picture_url VARCHAR(500),
        phone_number VARCHAR(11),
        provider_info VARCHAR(255),
        creation_date DATE, last_sign_in_date DATE
);

CREATE TABLE preferences (
        useindexr_id VARCHAR(500) PRIMARY KEY,
        zipcode VARCHAR(500),
        school VARCHAR(500),
```

```
    safety_level VARCHAR(500),
    hospital_proximity VARCHAR(500),
    recreation_proximity VARCHAR(500)
);
```

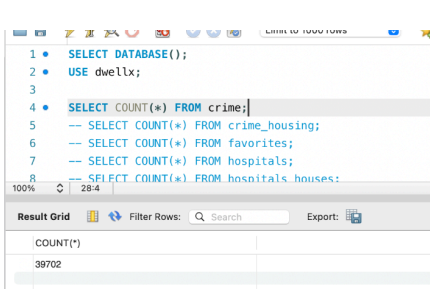
Intermediate Tables:

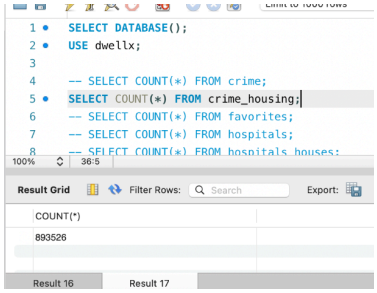
```
Unset
CREATE TABLE school_district (
    housing_id INT,
    school_id INT,
    distance DECIMAL(9,6),
    PRIMARY KEY (housing_id, school_id),
    FOREIGN KEY (housing_id) REFERENCES housing(index) ON
    DELETE CASCADE,
    FOREIGN KEY (school_id) REFERENCES schools(School_ID) ON
    DELETE CASCADE
);

CREATE TABLE hospitals_houses (
    housing_id INT,
    hospital_id INT,
    distance DECIMAL(9,6),
    PRIMARY KEY (housing_id, hospital_id),
```

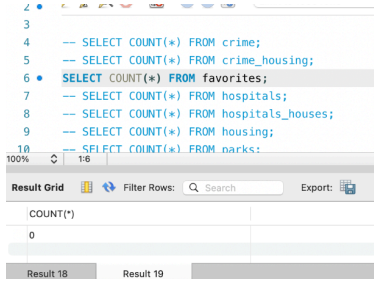
```
        FOREIGN KEY (housing_id) REFERENCES housing(index) ON
        DELETE CASCADE,
        FOREIGN KEY (hospital_id) REFERENCES hospitals(BLDGID) ON
        DELETE CASCADE
);

CREATE TABLE parks_houses (
        housing_id INT,
        park_id INT,
        distance DECIMAL(9,6),
        PRIMARY KEY (housing_id, park_id),
        FOREIGN KEY (housing_id) REFERENCES housing(index) ON
        DELETE CASCADE,
        FOREIGN KEY (park_id) REFERENCES parks(OBJECTID_1) ON DELETE
        CASCADE
);

CREATE TABLE crime_housing (
        case_number INT,
        housing_id INT,
        PRIMARY KEY (case_number, housing_id),
        FOREIGN KEY (case_number) REFERENCES crime(index) ON DELETE
        CASCADE,
        FOREIGN KEY (housing_id) REFERENCES housing(index) ON
        DELETE CASCADE
);
```

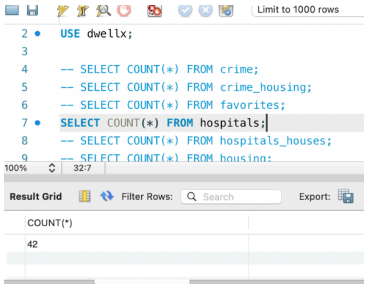
```
CREATE TABLE houses_favorited (
        housing_id INT,
        user_id VARCHAR(500),
        PRIMARY KEY (housing_id, user_id),
        FOREIGN KEY (housing_id) REFERENCES housing(index) ON
        DELETE CASCADE,
        FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE
        CASCADE
);
```

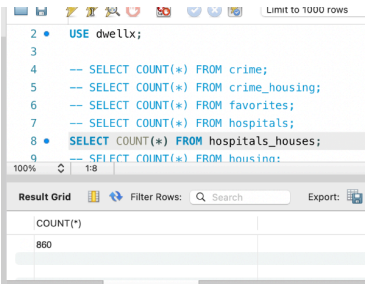
***Counts for the Tables*:**

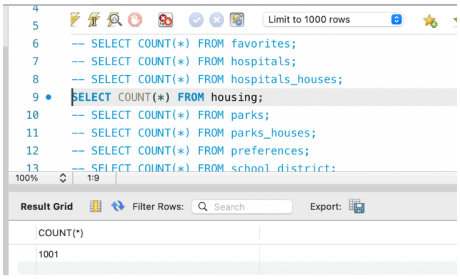


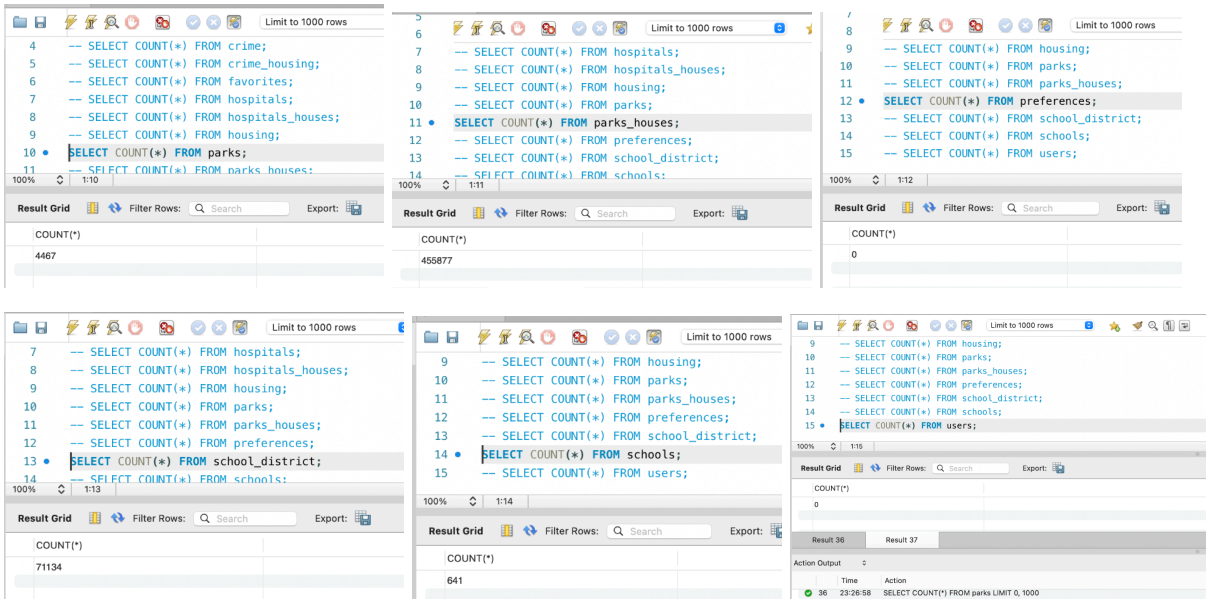












Advanced SQL Queries

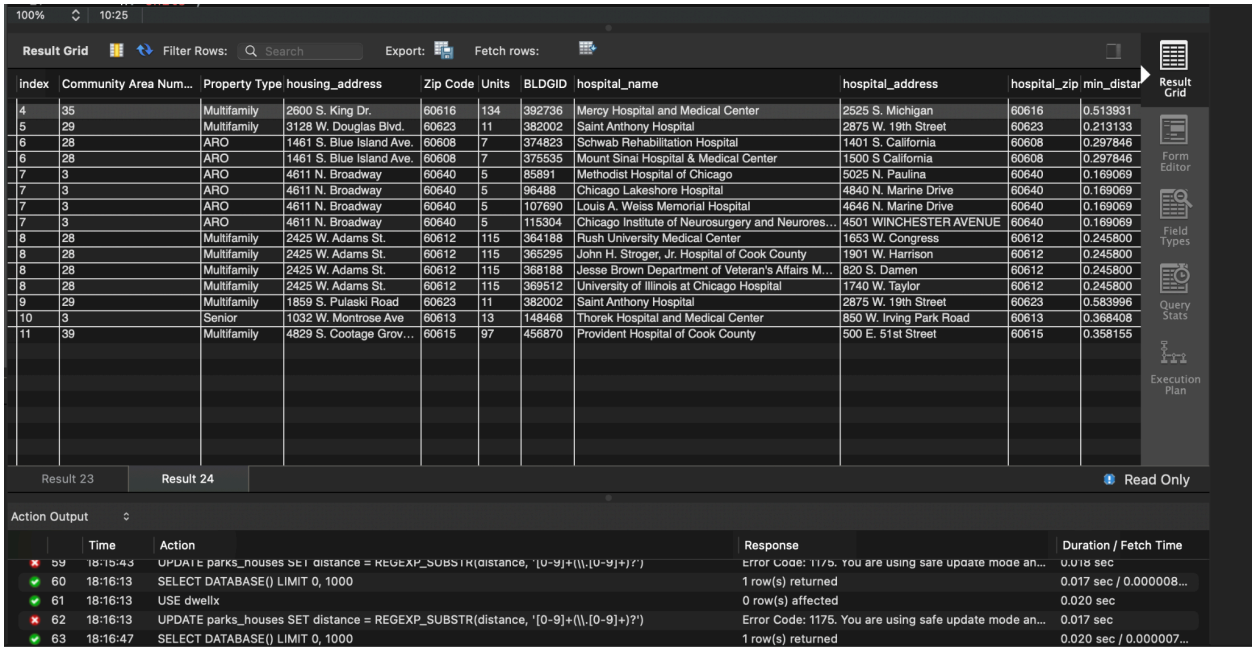
QUERY 1: Find housing units that are within 5km of a park and have a hospital nearby

Unset

```
SELECT DATABASE();
USE dwellx;
SELECT
    h.`index`,
    h.`Community Area Number`,
    h.`Property Type`,
    h.`Address` AS housing_address,
    h.`Zip Code`,
    h.`Units`,
    hosp.`BLDGID`,
    hosp.`FACILITY` AS hospital_name,
    hosp.`ADDRESS` AS hospital_address,
    hosp.`ZIP` AS hospital_zip,
    MIN(ph.`distance`) AS min_distance,
```

```
   GROUP_CONCAT(DISTINCT p.park ORDER BY ph.`distance` ASC
SEPARATOR ', ') AS park_names
FROM housing h
JOIN parks_houses ph ON h.`index` = ph.`housing_id`
JOIN parks p ON p.`OBJECTID_1` = ph.`park_id`
JOIN hospitals hosp ON h.`Zip Code` = hosp.`ZIP`
WHERE ph.`distance` <= 5
GROUP BY
   h.`index`,
   h.`Community Area Number`,
   h.`Property Type`,
   h.`Address`,
   h.`Zip Code`,
   h.`Units`,
   hosp.`BLDGID`,
   hosp.`FACILITY`,
   hosp.`ADDRESS`,
   hosp.`ZIP`
LIMIT 15;
```

OUTPUT:



Result Grid | Filter Rows: Search | Export: | Fetch rows:

index	Community Area Num...	Property Type	housing_address	Zip Code	Units	BLDGID	hospital_name	hospital_address	hospital_zip	min_distar
4	35	Multifamily	2600 S. King Dr.	60616	134	392736	Mercy Hospital and Medical Center	2525 S. Michigan	60616	0.513931
5	29	Multifamily	3128 W. Douglas Blvd.	60623	11	382002	Saint Anthony Hospital	2875 W. 19th Street	60623	0.213133
6	28	ARO	1461 S. Blue Island Ave.	60608	7	374823	Schwab Rehabilitation Hospital	1401 S. California	60608	0.297846
6	28	ARO	1461 S. Blue Island Ave.	60608	7	375535	Mount Sinai Hospital & Medical Center	1500 S California	60608	0.297846
7	3	ARO	4611 N. Broadway	60640	5	85891	Methodist Hospital of Chicago	5025 N. Paulina	60640	0.169069
7	3	ARO	4611 N. Broadway	60640	5	96488	Chicago Lakeshore Hospital	4840 N. Marine Drive	60640	0.169069
7	3	ARO	4611 N. Broadway	60640	5	107690	Louis A. Weiss Memorial Hospital	4646 N. Marine Drive	60640	0.169069
7	3	ARO	4611 N. Broadway	60640	5	115304	Chicago Institute of Neurosurgery and Neurores...	4501 WINCHESTER AVENUE	60640	0.169069
8	28	Multifamily	2425 W. Adams St.	60612	115	364188	Rush University Medical Center	1653 W. Congress	60612	0.245800
8	28	Multifamily	2425 W. Adams St.	60612	115	365295	John H. Stroger, Jr. Hospital of Cook County	1901 W. Harrison	60612	0.245800
8	28	Multifamily	2425 W. Adams St.	60612	115	368188	Jesse Brown Department of Veteran's Affairs M...	820 S. Damen	60612	0.245800
8	28	Multifamily	2425 W. Adams St.	60612	115	369512	University of Illinois at Chicago Hospital	1740 W. Taylor	60612	0.245800
9	29	Multifamily	1859 S. Pulaski Road	60623	11	382002	Saint Anthony Hospital	2875 W. 19th Street	60623	0.583996
10	3	Senior	1032 W. Montrose Ave	60613	13	148468	Thorek Hospital and Medical Center	850 W. Irving Park Road	60613	0.368408
11	39	Multifamily	4829 S. Cootage Grov...	60615	97	456870	Provident Hospital of Cook County	500 E. 51st Street	60615	0.358155

Result 23 | Result 24 | Read Only

Action Output

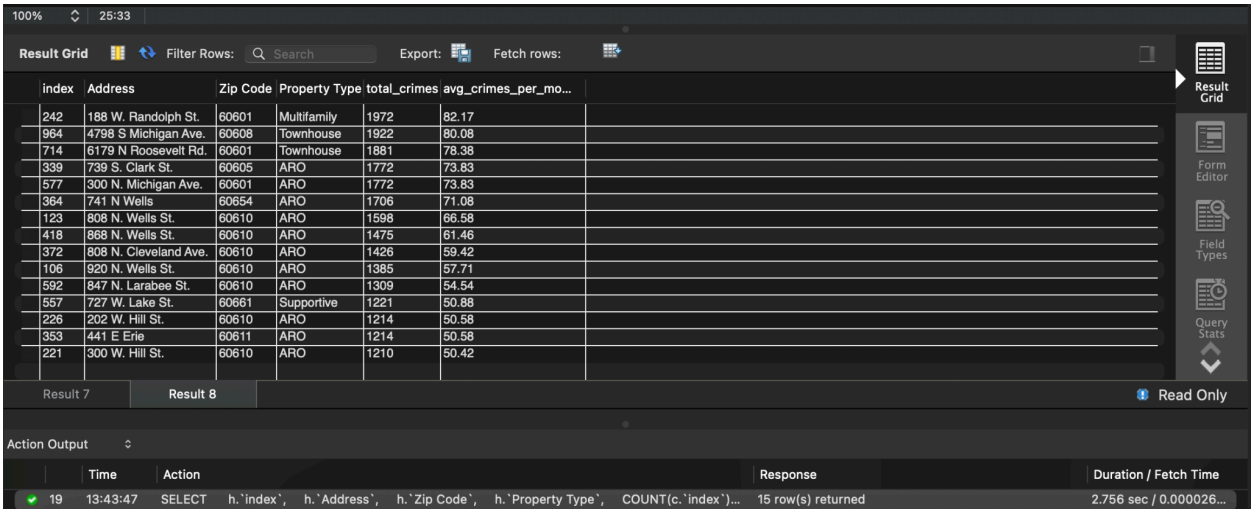
	Time	Action	Response	Duration / Fetch Time
59	18:15:43	UPDATE parks_houses SET distance = REGEXP_SUBSTR(distance, '[0-9]+(\\.[0-9]+)?')	Error Code: 1175. You are using safe update mode an...	0.018 sec
60	18:16:13	SELECT DATABASE() LIMIT 0, 1000	1 row(s) returned	0.017 sec / 0.000008...
61	18:16:13	USE dwellx	0 row(s) affected	0.020 sec
62	18:16:13	UPDATE parks_houses SET distance = REGEXP_SUBSTR(distance, '[0-9]+(\\.[0-9]+)?')	Error Code: 1175. You are using safe update mode an...	0.017 sec
63	18:16:47	SELECT DATABASE() LIMIT 0, 1000	1 row(s) returned	0.020 sec / 0.000007...

QUERY 2: **Find housing units with more than 3 crime cases reported within 1 mile**

```
Unset
SELECT DATABASE();
USE dwellx;
EXPLAIN ANALYZE SELECT
  h.`index`,
  h.`Address`,
  h.`Zip Code`,
  h.`Property Type`,
  COUNT(c.`index`) AS total_crimes,
  ROUND(COUNT(c.`index`) / 24.0, 2) AS avg_crimes_per_month
FROM
  housing h
JOIN
  crime_housing ch ON h.`index` = ch.`housing_id`
JOIN
  crime c ON ch.`case_number` = c.`index`
WHERE
  (6371 * ACOS(
    COS(RADIANS(c.`LATITUDE`)) * COS(RADIANS(h.`Latitude`)) *
    COS(RADIANS(h.`Longitude`) - RADIANS(c.`LONGITUDE`)) +
    SIN(RADIANS(c.`LATITUDE`)) * SIN(RADIANS(h.`Latitude`))
  )) <= 1.609
  AND (
    c.`iucr` LIKE '0%' OR
    c.`iucr` LIKE '%ASSAULT%' OR
    c.`iucr` LIKE '%BATTERY%' OR
    c.`iucr` LIKE '%HOMICIDE%' OR
    c.`iucr` LIKE '%ROBBERY%' OR
    c.`iucr` LIKE '%WEAPON%'
  )
GROUP BY
  h.`index`, h.`Address`, h.`Zip Code`, h.`Property Type`
HAVING
  COUNT(c.`index`) > 3
```

```
ORDER BY
   total_crimes DESC;
LIMIT 15;
```

OUTPUT:



index	Address	Zip Code	Property Type	total_crimes	avg_crimes_per_mo...
242	188 W. Randolph St.	60601	Multifamily	1972	82.17
964	4798 S Michigan Ave.	60608	Townhouse	1922	80.08
714	6179 N Roosevelt Rd.	60601	Townhouse	1881	78.38
339	739 S. Clark St.	60605	ARO	1772	73.83
577	300 N. Michigan Ave.	60601	ARO	1772	73.83
364	741 N Wells	60654	ARO	1706	71.08
123	808 N. Wells St.	60610	ARO	1598	66.58
418	868 N. Wells St.	60610	ARO	1475	61.46
372	808 N. Cleveland Ave.	60610	ARO	1426	59.42
106	920 N. Wells St.	60610	ARO	1385	57.71
592	847 N. Larabee St.	60610	ARO	1309	54.54
557	727 W. Lake St.	60661	Supportive	1221	50.88
226	202 W. Hill St.	60610	ARO	1214	50.58
353	441 E Erie	60611	ARO	1214	50.58
221	300 W. Hill St.	60610	ARO	1210	50.42

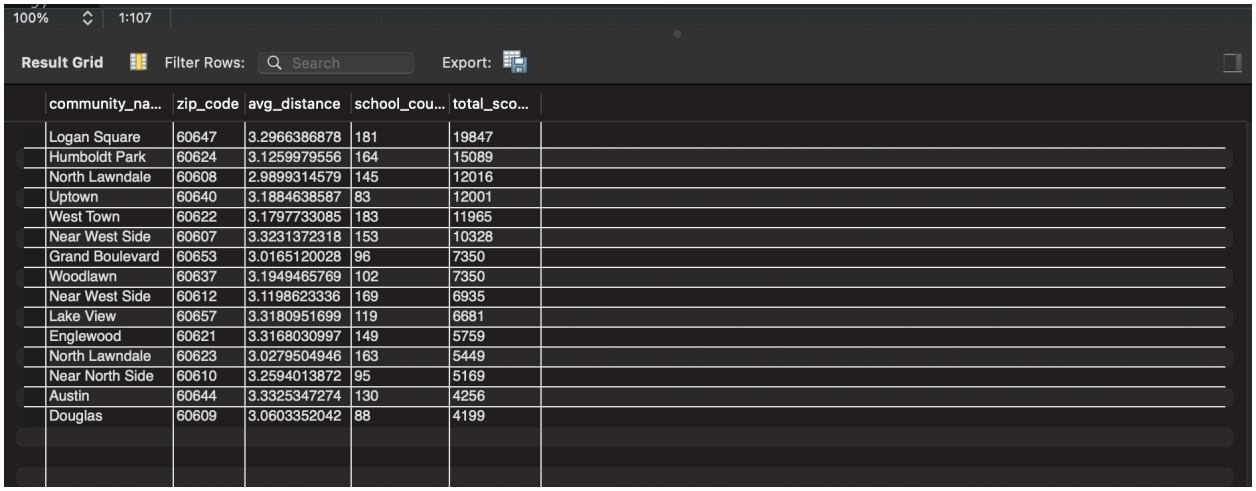
QUERY 3: **Find the best housing communities according to the closest school rating and average school distance**

```
Unset
WITH RatedSchools AS (
 SELECT
   s.school_id,
   sd.housing_id,
   h.`Community Area Name` AS community_name,
   h.`Zip Code` AS zip_code,
   sd.distance,

   CASE
    WHEN s.culture_climate_rating = 'WELL ORGANIZED' THEN 3
```

```
    WHEN s.culture_climate_rating = 'ORGANIZED' THEN 2
    ELSE 1
  END AS climate_score,

  CASE
    WHEN s.student_attainment_rating = 'NO DATA AVAILABLE' THEN 1
    ELSE 0
  END AS attainment_flag,

  CASE
    WHEN s.student_growth_rating = 'NO DATA AVAILABLE' THEN 1
    ELSE 0
  END AS growth_flag

 FROM schools s
 JOIN school_district sd ON s.school_id = sd.school_id
 JOIN housing h ON sd.housing_id = h.index
),

ScoredCommunities AS (
 SELECT
  community_name,
  zip_code,
  AVG(distance) AS avg_distance,
  COUNT(DISTINCT school_id) AS school_count,
  (SUM(climate_score) * 2 + SUM(attainment_flag + growth_flag)) AS
total_score

 FROM RatedSchools
 GROUP BY community_name, zip_code
)

SELECT sc.*
FROM ScoredCommunities sc
JOIN (
```

```
    -- Get the highest total_score for each zip code
    SELECT zip_code, MAX(total_score) AS max_score
    FROM ScoredCommunities
    GROUP BY zip_code
) max_scores
ON sc.zip_code = max_scores.zip_code AND sc.total_score =
max_scores.max_score
ORDER BY total_score DESC, avg_distance ASC
LIMIT 15;
```

OUTPUT:



community_na...	zip_code	avg_distance	school_cou...	total_sco...
Logan Square	60647	3.2966386878	181	19847
Humboldt Park	60624	3.1259979556	164	15089
North Lawndale	60608	2.9899314579	145	12016
Uptown	60640	3.1884638587	83	12001
West Town	60622	3.1797733085	183	11965
Near West Side	60607	3.3231372318	153	10328
Grand Boulevard	60653	3.0165120028	96	7350
Woodlawn	60637	3.1949465769	102	7350
Near West Side	60612	3.1198623336	169	6935
Lake View	60657	3.3180951699	119	6681
Englewood	60621	3.3168030997	149	5759
North Lawndale	60623	3.0279504946	163	5449
Near North Side	60610	3.2594013872	95	5169
Austin	60644	3.3325347274	130	4256
Douglas	60609	3.0603352042	88	4199

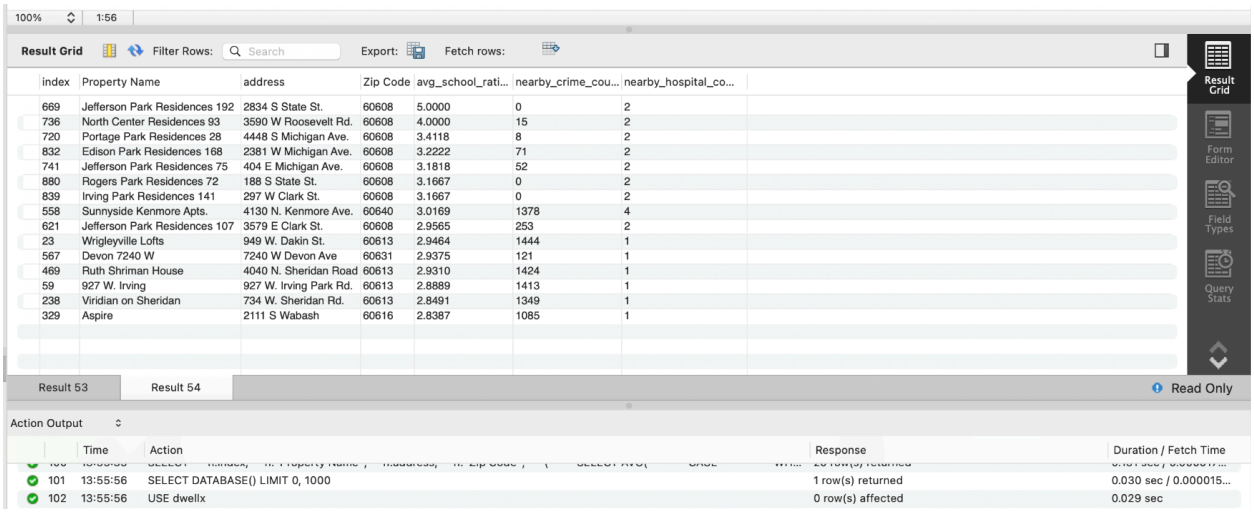
QUERY 4: **Find the best housing communities according to the closest school rating and average school distance**

```
Unset
SELECT
    h.index,
    h.`Property Name`,
    h.address,
    h.`Zip Code`,
```

```
  (
    SELECT AVG(
      CASE
        WHEN s.culture_climate_rating = 'WELL ORGANIZED' THEN 5
        WHEN s.culture_climate_rating = 'ORGANIZED' THEN 4
        WHEN s.culture_climate_rating = 'NOT YET ORGANIZED' THEN 2
        WHEN s.culture_climate_rating = 'NOT ENOUGH DATA' THEN 1
        ELSE 0
      END
    )

    FROM schools s
    JOIN school_district sd ON s.school_id = sd.school_id
    WHERE sd.housing_id = h.index
  ) AS avg_school_rating,

  (
    SELECT COUNT(*)
    FROM crime_housing ch
    WHERE ch.housing_id = h.index
  ) AS nearby_crime_count,

  (
    SELECT COUNT(*)
    FROM hospitals hosp
    WHERE hosp.zip = h.`Zip Code`
  ) AS nearby_hospital_count

FROM housing h

WHERE (
  SELECT COUNT(*)
  FROM hospitals hosp
  WHERE hosp.zip = h.`Zip Code`
) > 0

HAVING
```

```
    nearby_crime_count <= 1500
    AND avg_school_rating >= 2

ORDER BY
    avg_school_rating DESC,
    nearby_crime_count ASC,
    nearby_hospital_count DESC

LIMIT 15;
```

OUTPUT:



index	Property Name	address	Zip Code	avg_school_rati...	nearby_crime_cou...	nearby_hospital_co...
669	Jefferson Park Residences 192	2834 S State St.	60608	5.0000	0	2
736	North Center Residences 93	3590 W Roosevelt Rd.	60608	4.0000	15	2
720	Portage Park Residences 28	4448 S Michigan Ave.	60608	3.4118	8	2
832	Edison Park Residences 168	2381 W Michigan Ave.	60608	3.2222	71	2
741	Jefferson Park Residences 75	404 E Michigan Ave.	60608	3.1818	52	2
880	Rogers Park Residences 72	188 S State St.	60608	3.1667	0	2
839	Irving Park Residences 141	297 W Clark St.	60608	3.1667	0	2
558	Sunnyside Kenmore Apts.	4130 N. Kenmore Ave.	60640	3.0169	1378	4
621	Jefferson Park Residences 107	3579 E Clark St.	60608	2.9565	253	2
23	Wrigleyville Lofts	949 W. Dakin St.	60613	2.9464	1444	1
567	Devon 7240 W	7240 W Devon Ave	60631	2.9375	121	1
469	Ruth Shriman House	4040 N. Sheridan Road	60613	2.9310	1424	1
59	927 W. Irving	927 W. Irving Park Rd.	60613	2.8889	1413	1
238	Viridian on Sheridan	734 W. Sheridan Rd.	60613	2.8491	1349	1
329	Aspire	2111 S Wabash	60616	2.8387	1085	1

QUERY 5: **Find the area-wise aggregate insights by finding the number of parks, schools, and hospitals within or near a zip code**

```
Unset
SELECT
    h.`Zip Code`,
    SUM(ph.num_parks) AS num_parks,
    SUM(sd.num_schools) AS num_schools,
    SUM(hosp.num_hospitals) AS num_hospitals,
```

```
   SUM(ph.num_parks) + SUM(sd.num_schools) + SUM(hosp.num_hospitals) AS
total_count
FROM housing h
LEFT JOIN (
   SELECT housing_id, COUNT(DISTINCT park_id) AS num_parks
   FROM parks_houses
   WHERE distance <= 8
   GROUP BY housing_id
) ph ON h.index = ph.housing_id
LEFT JOIN (
   SELECT housing_id, COUNT(DISTINCT school_id) AS num_schools
   FROM school_district
   WHERE distance <= 5
   GROUP BY housing_id
) sd ON h.index = sd.housing_id
LEFT JOIN (
   SELECT ZIP, COUNT(DISTINCT BLDGID) AS num_hospitals
   FROM hospitals
   GROUP BY ZIP
) hosp ON h.`Zip Code` = hosp.ZIP
GROUP BY h.`Zip Code`
ORDER BY total_count DESC
LIMIT 15;
```

OUTPUT:

```
26    ORDER BY total_count DESC
27    LIMIT 15;
28
29
```

100% 1:28

Result Grid Filter Rows: Search Export: Fetch rows:

Zip Code	num_parks	num_schools	num_hospitals	total_count
60608	37666	6719	180	44565
60624	21865	5201	108	27174
60640	23055	2323	156	25534
60637	22578	2839	31	25448
60612	19181	4264	120	23565
60622	15252	2769	72	18093
60615	11964	1434	18	13416
60657	10097	1236	34	11367
60621	7960	1253	11	9224
60613	6846	868	12	7726
60623	6107	1576	13	7696
60616	6655	784	12	7451
60641	4911	730	8	5649
60628	4581	639	11	5231
60660	4512	417	8	4937

Result Grid | Form Editor | Field Types | Query Stats

Result 20 Read Only

Action Output

	Time	Action	Response	Duration / Fetch Time
49	18:27:21	SELECT h.`Zip Code`, COALESCE(ph.num_parks, 0) AS num_parks, COALESCE(sd.num_schools, 0) AS num_school...	Error Code: 1055. Expression #2 of SELECT list is not...	0.029 sec
50	18:28:48	SELECT h.`Zip Code`, COALESCE(ph.num_parks, 0) AS num_parks, COALESCE(sd.num_schools, 0) AS num_school...	Error Code: 1055. Expression #2 of SELECT list is not...	0.031 sec

Indexing and Analysis

QUERY 1:

ORIGINAL

```
+------------------------------------------------------------------------------------------------------------+
| -> Group aggregate: min(parks_houses.distance), group_concat(distinct parks.park order by parks_houses.distance ASC separator ', ')  (actual time=3003..3786
rows=834 loops=1)
    -> Sort: h.`index`, h.`Community Area Number`, h.`Property Type`, h.Address, h.`Zip Code`, h.Units, hosp.BLDGID, hosp.FACILITY, hosp.ADDRESS, hosp.ZIP  (ac
tual time=3002..3174 rows=483529 loops=1)
        -> Stream results  (cost=887563 rows=764106) (actual time=7.34..2118 rows=483529 loops=1)
            -> Nested loop inner join  (cost=887563 rows=764106) (actual time=7.32..1729 rows=483529 loops=1)
                -> Nested loop inner join  (cost=806604 rows=764106) (actual time=7.29..1197 rows=483529 loops=1)
                    -> Inner hash join (h.`Zip Code` = hosp.ZIP)  (cost=4213 rows=4204) (actual time=0.0955..3.93 rows=860 loops=1)
                        -> Table scan on h  (cost=0.331 rows=1001) (actual time=0.0165..1.86 rows=1001 loops=1)
                        -> Hash
                            -> Table scan on hosp  (cost=4.45 rows=42) (actual time=0.0333..0.0562 rows=42 loops=1)
                    -> Filter: (ph.distance <= 5.000000)  (cost=136 rows=182) (actual time=0.508..1.35 rows=562 loops=860)
                        -> Index lookup on ph using housing_id (housing_id=h.`index`)  (cost=136 rows=545) (actual time=0.507..1.29 rows=562 loops=860)
                -> Single-row index lookup on p using PRIMARY (OBJECTID_1=ph.park_id)  (cost=0.00595 rows=1) (actual time=914e-6..943e-6 rows=1 loops=483529)
 |
+------------------------------------------------------------------------------------------------------------+
1 row in set, 717 warnings (3.81 sec)
```

```
Unset
CREATE INDEX zipcode ON hospitals (`ZIP`);
```

```
+------------------------------------------------------------------------------------------------------------+
| -> Group aggregate: min(parks_houses.distance), group_concat(distinct parks.park order by parks_houses.distance ASC separator ', ')  (actual time=3028..3791
rows=834 loops=1)
    -> Sort: h.`index`, h.`Community Area Number`, h.`Property Type`, h.Address, h.`Zip Code`, h.Units, hosp.BLDGID, hosp.FACILITY, hosp.ADDRESS, hosp.ZIP  (ac
tual time=3027..3194 rows=483529 loops=1)
        -> Stream results  (cost=212335 rows=285829) (actual time=0.102..2001 rows=483529 loops=1)
            -> Nested loop inner join  (cost=212335 rows=285829) (actual time=0.0978..1605 rows=483529 loops=1)
                -> Nested loop inner join  (cost=112295 rows=170137) (actual time=0.0743..759 rows=455877 loops=1)
                    -> Nested loop inner join  (cost=52747 rows=170137) (actual time=0.0621..199 rows=455877 loops=1)
                        -> Table scan on p  (cost=455 rows=4481) (actual time=0.0242..2.8 rows=4467 loops=1)
                        -> Filter: (ph.distance <= 5.000000)  (cost=0.279 rows=38) (actual time=0.0115..0.0376 rows=102 loops=4467)
                            -> Index lookup on ph using PRIMARY (park_id=p.OBJECTID_1)  (cost=0.279 rows=114) (actual time=0.0113..0.0255 rows=102 loops=4467)
                    -> Filter: (h.`Zip Code` is not null)  (cost=0.25 rows=1) (actual time=984e-6..0.00107 rows=1 loops=455877)
                        -> Single-row index lookup on h using PRIMARY (index=ph.housing_id)  (cost=0.25 rows=1) (actual time=852e-6..881e-6 rows=1 loops=455877
)
                -> Index lookup on hosp using zipcode (ZIP=h.`Zip Code`)  (cost=0.42 rows=1.68) (actual time=969e-6..0.00167 rows=1.06 loops=455877)
 |
+------------------------------------------------------------------------------------------------------------+
1 row in set, 717 warnings (3.81 sec)
```

Unset

```
CREATE INDEX unit ON housing (`UNIT`);
```

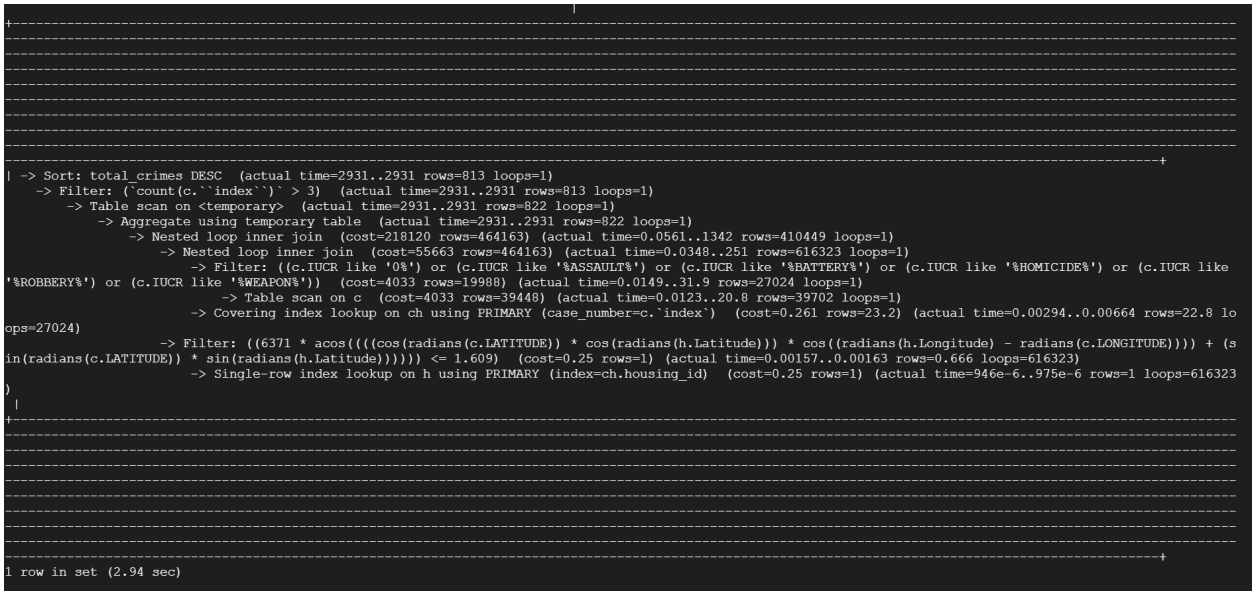
Unset

```
CREATE INDEX dist ON parks_houses (`distance`);
```

With our first query, our cost was 887563 without using any of the indexing. For our first indexing design, we decided to add an index to the ZIP column for hospitals. This improved our cost by reducing it to 212335. For the second indexing design, we added to keep the ZIP indexing for hospitals and also added indexing for the distance column from the parks_houses table. This increased the cost to 292153. For our third indexing design, we decided to keep the previous two indexes and add an index to the Units column in the housing table. This didn't change the cost of the query and kept it at 292153. In the end, we decided to choose our first indexing design, as it significantly reduced the cost from 887563 to 212335. We believe that the cost was significantly reduced because the ZIP column is used in a lot of portions of the query, so it reduced the table lookup time by a lot.

QUERY 2:

ORIGINAL



```
| -> Sort: total_crimes DESC  (actual time=2931..2931 rows=813 loops=1)
    -> Filter: (`count(c.``index``)` > 3)  (actual time=2931..2931 rows=813 loops=1)
        -> Table scan on <temporary>  (actual time=2931..2931 rows=822 loops=1)
            -> Aggregate using temporary table  (actual time=2931..2931 rows=822 loops=1)
                -> Nested loop inner join  (cost=218120 rows=464163) (actual time=0.0561..1342 rows=410449 loops=1)
                    -> Nested loop inner join  (cost=55663 rows=464163) (actual time=0.0348..251 rows=616323 loops=1)
                        -> Filter: ((c.IUCR like '0%') or (c.IUCR like '%ASSAULT%') or (c.IUCR like '%BATTERY%') or (c.IUCR like '%HOMICIDE%') or (c.IUCR like
'%ROBBERY%') or (c.IUCR like '%WEAPON%'))  (cost=4033 rows=19988) (actual time=0.0149..31.9 rows=27024 loops=1)
                            -> Table scan on c  (cost=4033 rows=39448) (actual time=0.0123..20.8 rows=39702 loops=1)
                        -> Covering index lookup on ch using PRIMARY (case_number=c.`index`)  (cost=0.261 rows=23.2) (actual time=0.00294..0.00664 rows=22.8 lo
ops=27024)
                    -> Filter: ((6371 * acos((((cos(radians(c.LATITUDE)) * cos(radians(h.Latitude))) * cos((radians(h.Longitude) - radians(c.LONGITUDE)))) + (s
in(radians(c.LATITUDE)) * sin(radians(h.Latitude))))) <= 1.609)  (cost=0.25 rows=1) (actual time=0.00157..0.00163 rows=0.666 loops=616323)
                        -> Single-row index lookup on h using PRIMARY (index=ch.housing_id)  (cost=0.25 rows=1) (actual time=946e-6..975e-6 rows=1 loops=616323
)
 |
1 row in set (2.94 sec)
```

Unset

```
CREATE INDEX zipcode ON housing(`ZIP CODE`);
```

```
+----------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| -> Sort: total_crimes DESC  (actual time=3025..3025 rows=813 loops=1)
    -> Filter: (`count(c.``index``)` > 3)  (actual time=3024..3025 rows=813 loops=1)
        -> Table scan on <temporary>  (actual time=3024..3025 rows=822 loops=1)
            -> Aggregate using temporary table  (actual time=3024..3024 rows=822 loops=1)
                -> Nested loop inner join  (cost=218120 rows=464163) (actual time=0.0465..1399 rows=410449 loops=1)
                    -> Nested loop inner join  (cost=55663 rows=464163) (actual time=0.0278..287 rows=616323 loops=1)
                        -> Filter: ((c.IUCR like '0%') or (c.IUCR like '%ASSAULT%') or (c.IUCR like '%BATTERY%') or (c.IUCR like '%HOMICIDE%') or (c.IUCR like
'%ROBBERY%') or (c.IUCR like '%WEAPON%'))  (cost=4033 rows=19988) (actual time=0.0145..33.2 rows=27024 loops=1)
                            -> Table scan on c  (cost=4033 rows=39448) (actual time=0.0113..21.7 rows=39702 loops=1)
                        -> Covering index lookup on ch using PRIMARY (case_number=c.`index`)  (cost=0.261 rows=23.2) (actual time=0.0041..0.00789 rows=22.8 loo
ps=27024)
                    -> Filter: ((6371 * acos((((cos(radians(c.LATITUDE)) * cos(radians(h.Latitude))) * cos((radians(h.Longitude) - radians(c.LONGITUDE)))) + (s
in(radians(c.LATITUDE)) * sin(radians(h.Latitude)))))) <= 1.609)  (cost=0.25 rows=1) (actual time=0.00161..0.00167 rows=0.666 loops=616323)
                        -> Single-row index lookup on h using PRIMARY (index=ch.housing_id)  (cost=0.25 rows=1) (actual time=974e-6..0.001 rows=1 loops=616323)
 |
+----------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
1 row in set (3.03 sec)
```

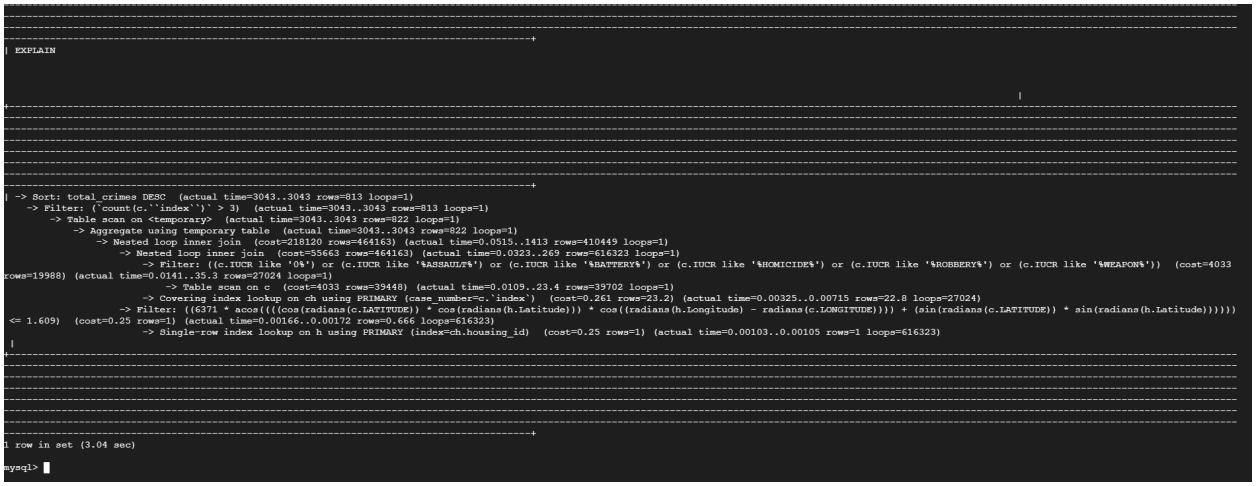
Unset

```
CREATE INDEX property_type ON housing(`Property Type`);
```

```
+----------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| -> Sort: total_crimes DESC  (actual time=3019..3019 rows=813 loops=1)
    -> Filter: (`count(c.``index``)` > 3)  (actual time=3019..3019 rows=813 loops=1)
        -> Table scan on <temporary>  (actual time=3019..3019 rows=822 loops=1)
            -> Aggregate using temporary table  (actual time=3019..3019 rows=822 loops=1)
                -> Nested loop inner join  (cost=218120 rows=464163) (actual time=0.049..1403 rows=410449 loops=1)
                    -> Nested loop inner join  (cost=55663 rows=464163) (actual time=0.0305..265 rows=616323 loops=1)
                        -> Filter: ((c.IUCR like '0%') or (c.IUCR like '%ASSAULT%') or (c.IUCR like '%BATTERY%') or (c.IUCR like '%HOMICIDE%') or (c.IUCR like
'%ROBBERY%') or (c.IUCR like '%WEAPON%'))  (cost=4033 rows=19988) (actual time=0.0144..33.7 rows=27024 loops=1)
                            -> Table scan on c  (cost=4033 rows=39448) (actual time=0.0111..21.7 rows=39702 loops=1)
                        -> Covering index lookup on ch using PRIMARY (case_number=c.`index`)  (cost=0.261 rows=23.2) (actual time=0.00327..0.00706 rows=22.8 lo
ops=27024)
                    -> Filter: ((6371 * acos((((cos(radians(c.LATITUDE)) * cos(radians(h.Latitude))) * cos((radians(h.Longitude) - radians(c.LONGITUDE)))) + (s
in(radians(c.LATITUDE)) * sin(radians(h.Latitude)))))) <= 1.609)  (cost=0.25 rows=1) (actual time=0.00165..0.00171 rows=0.666 loops=616323)
                        -> Single-row index lookup on h using PRIMARY (index=ch.housing_id)  (cost=0.25 rows=1) (actual time=0.00102..0.00105 rows=1 loops=6163
23)
 |
+----------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
1 row in set (3.03 sec)
```

Unset

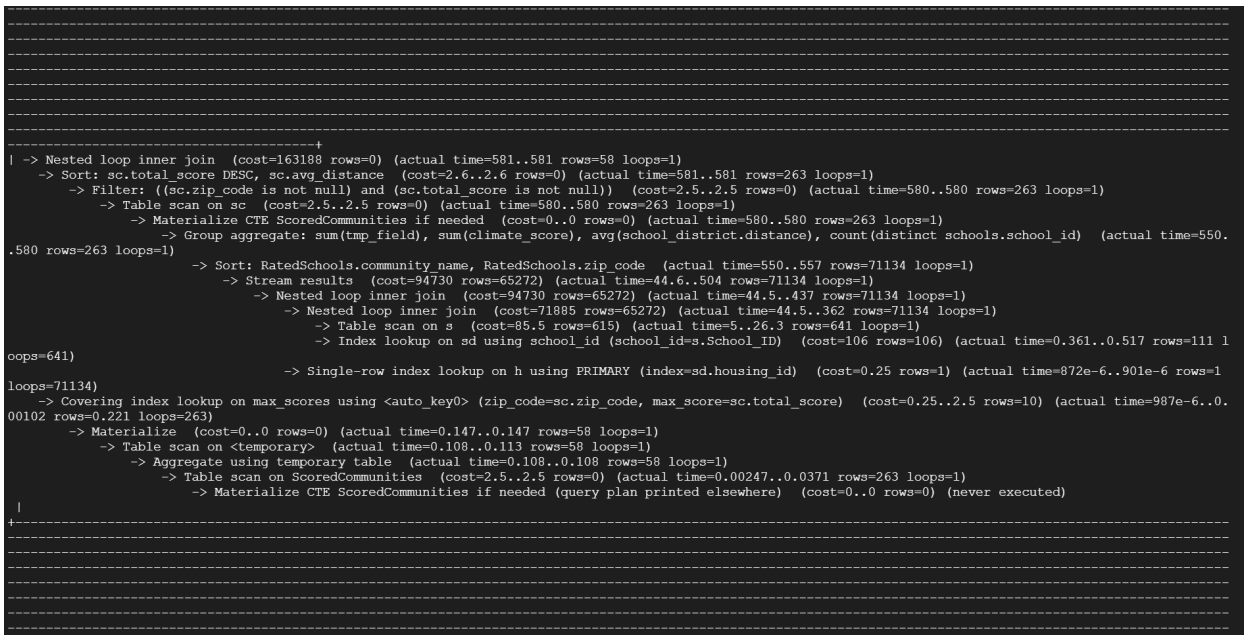
```
CREATE INDEX addr ON housing (`Address`(255));
```



The original query had a cost of 218120 before indexing. The columns we identified as potential indexes are `Zip Code` and `Property Type`, as the other attributes used in the HAVING, GROUP BY, and JOIN clauses are Primary Keys. After indexing the `Zip Code` attribute of the housing entity, the cost remained unchanged at 218120. Similarly, when we created an index on the `Property Type` column, the cost stayed the same. For the third design, we added an index on the `Address` column from the housing table. After indexing, the cost still remained unchanged at 218120. Most of the cost incurred by the query was a result of the inner joins using crime, housing, and the intermediate table crime_housing. I think this change in indexing did not create much of a difference in the overall cost, as these attributes were not a part of the joins or filtering in a meaningful way. Since there was no change, we decided to stick with the original query to reduce the added overhead of maintaining the indexes.

QUERY 3:

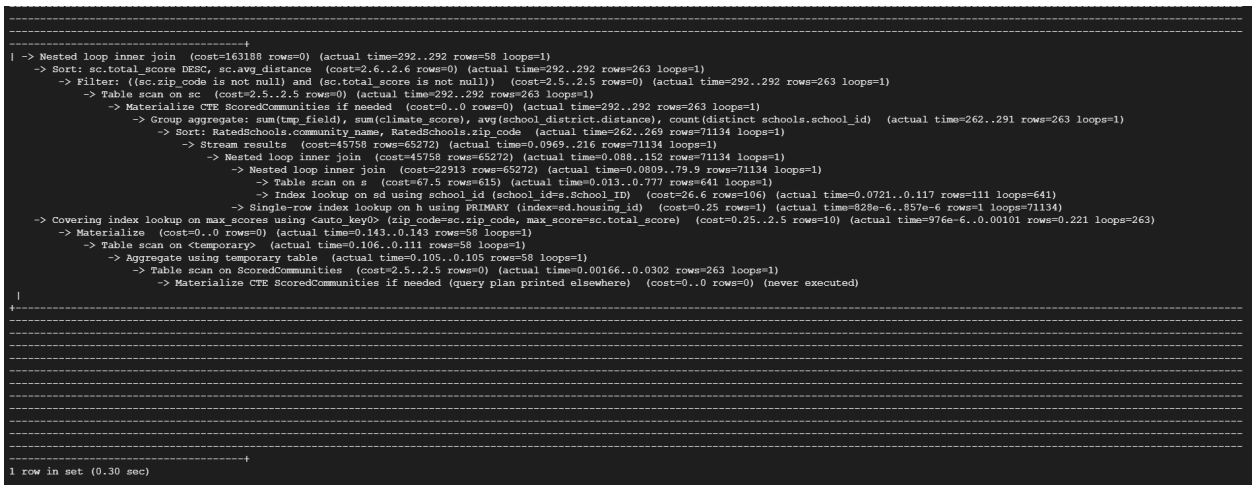
ORIGINAL



```
| -> Nested loop inner join  (cost=163188 rows=0) (actual time=581..581 rows=58 loops=1)
    -> Sort: sc.total_score DESC, sc.avg_distance  (cost=2.6..2.6 rows=0) (actual time=581..581 rows=263 loops=1)
        -> Filter: ((sc.zip_code is not null) and (sc.total_score is not null))  (cost=2.5..2.5 rows=0) (actual time=580..580 rows=263 loops=1)
            -> Table scan on sc  (cost=2.5..2.5 rows=0) (actual time=580..580 rows=263 loops=1)
                -> Materialize CTE ScoredCommunities if needed  (cost=0..0 rows=0) (actual time=580..580 rows=263 loops=1)
                    -> Group aggregate: sum(tmp_field), sum(climate_score), avg(school_district.distance), count(distinct schools.school_id)  (actual time=550.
.580 rows=263 loops=1)
                        -> Sort: RatedSchools.community_name, RatedSchools.zip_code  (actual time=550..557 rows=71134 loops=1)
                            -> Stream results  (cost=94730 rows=65272) (actual time=44.6..504 rows=71134 loops=1)
                                -> Nested loop inner join  (cost=94730 rows=65272) (actual time=44.5..437 rows=71134 loops=1)
                                    -> Nested loop inner join  (cost=71885 rows=65272) (actual time=44.5..362 rows=71134 loops=1)
                                        -> Table scan on s  (cost=85.5 rows=615) (actual time=5..26.3 rows=641 loops=1)
                                        -> Index lookup on sd using school_id (school_id=s.School_ID)  (cost=106 rows=106) (actual time=0.361..0.517 rows=111 l
oops=641)
                                    -> Single-row index lookup on h using PRIMARY (index=sd.housing_id)  (cost=0.25 rows=1) (actual time=872e-6..901e-6 rows=1
loops=71134)
    -> Covering index lookup on max_scores using <auto_key0> (zip_code=sc.zip_code, max_score=sc.total_score)  (cost=0.25..2.5 rows=10) (actual time=987e-6..0.
00102 rows=0.221 loops=263)
        -> Materialize  (cost=0..0 rows=0) (actual time=0.147..0.147 rows=58 loops=1)
            -> Table scan on <temporary>  (actual time=0.108..0.113 rows=58 loops=1)
                -> Aggregate using temporary table  (actual time=0.108..0.108 rows=58 loops=1)
                    -> Table scan on ScoredCommunities  (cost=2.5..2.5 rows=0) (actual time=0.00247..0.0371 rows=263 loops=1)
                        -> Materialize CTE ScoredCommunities if needed (query plan printed elsewhere)  (cost=0..0 rows=0) (never executed)
 |
```

Unset

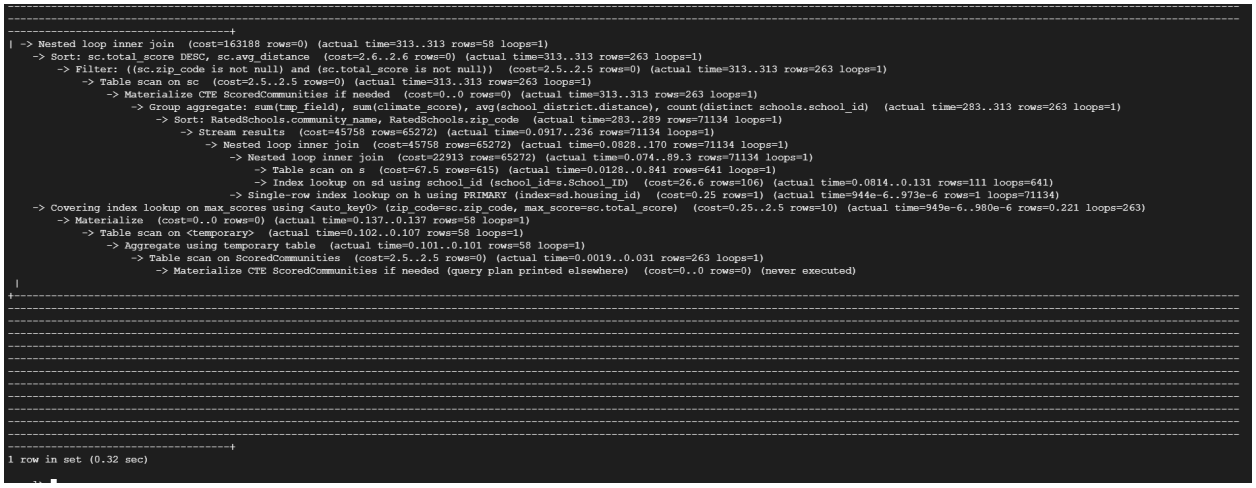
```
CREATE INDEX zipcode ON housing(`Zip Code`);
```



```
| -> Nested loop inner join  (cost=163188 rows=0) (actual time=292..292 rows=58 loops=1)
    -> Sort: sc.total_score DESC, sc.avg_distance  (cost=2.6..2.6 rows=0) (actual time=292..292 rows=263 loops=1)
        -> Filter: ((sc.zip_code is not null) and (sc.total_score is not null))  (cost=2.5..2.5 rows=0) (actual time=292..292 rows=263 loops=1)
            -> Table scan on sc  (cost=2.5..2.5 rows=0) (actual time=292..292 rows=263 loops=1)
                -> Materialize CTE ScoredCommunities if needed  (cost=0..0 rows=0) (actual time=292..292 rows=263 loops=1)
                    -> Group aggregate: sum(tmp_field), sum(climate_score), avg(school_district.distance), count(distinct schools.school_id)  (actual time=262..291 rows=263 loops=1)
                        -> Sort: RatedSchools.community_name, RatedSchools.zip_code  (actual time=262..269 rows=71134 loops=1)
                            -> Stream results  (cost=45758 rows=65272) (actual time=0.0969..216 rows=71134 loops=1)
                                -> Nested loop inner join  (cost=45758 rows=65272) (actual time=0.088..152 rows=71134 loops=1)
                                    -> Nested loop inner join  (cost=22913 rows=65272) (actual time=0.0809..79.9 rows=71134 loops=1)
                                        -> Table scan on s  (cost=67.5 rows=615) (actual time=0.013..0.777 rows=641 loops=1)
                                        -> Index lookup on sd using school_id (school_id=s.School_ID)  (cost=26.6 rows=106) (actual time=0.0721..0.117 rows=111 loops=641)
                                    -> Single-row index lookup on h using PRIMARY (index=sd.housing_id)  (cost=0.25 rows=1) (actual time=828e-6..857e-6 rows=1 loops=71134)
    -> Covering index lookup on max_scores using <auto_key0> (zip_code=sc.zip_code, max_score=sc.total_score)  (cost=0.25..2.5 rows=10) (actual time=976e-6..0.00101 rows=0.221 loops=263)
        -> Materialize  (cost=0..0 rows=0) (actual time=0.143..0.143 rows=58 loops=1)
            -> Table scan on <temporary>  (actual time=0.106..0.111 rows=58 loops=1)
                -> Aggregate using temporary table  (actual time=0.105..0.105 rows=58 loops=1)
                    -> Table scan on ScoredCommunities  (cost=2.5..2.5 rows=0) (actual time=0.00166..0.0302 rows=263 loops=1)
                        -> Materialize CTE ScoredCommunities if needed (query plan printed elsewhere)  (cost=0..0 rows=0) (never executed)
 |
1 row in set (0.30 sec)
```

Unset

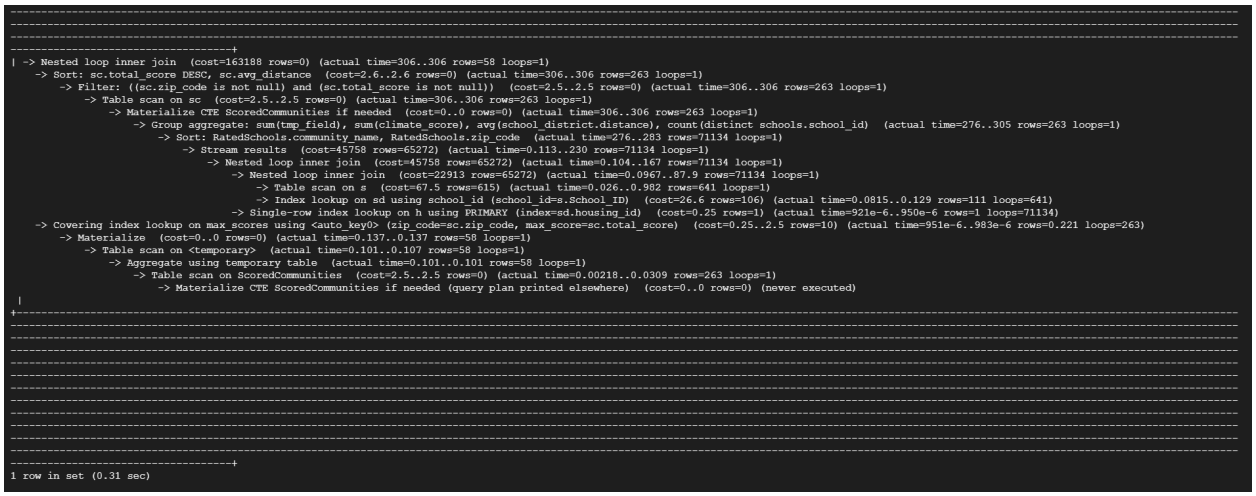
```
CREATE INDEX comm_name ON housing(`Community Area Name`(255));
```



```
-> Nested loop inner join  (cost=163188 rows=0) (actual time=313..313 rows=58 loops=1)
    -> Sort: sc.total_score DESC, sc.avg_distance  (cost=2.6..2.6 rows=0) (actual time=313..313 rows=263 loops=1)
        -> Filter: ((sc.zip_code is not null) and (sc.total_score is not null))  (cost=2.5..2.5 rows=0) (actual time=313..313 rows=263 loops=1)
            -> Table scan on sc  (cost=2.5..2.5 rows=0) (actual time=313..313 rows=263 loops=1)
                -> Materialize CTE ScoredCommunities if needed  (cost=0..0 rows=0) (actual time=313..313 rows=263 loops=1)
                    -> Group aggregate: sum(tmp_field), sum(climate_score), avg(school_district.distance), count(distinct schools.school_id)  (actual time=283..313 rows=263 loops=1)
                        -> Sort: RatedSchools.community_name, RatedSchools.zip_code  (actual time=283..289 rows=71134 loops=1)
                            -> Stream results  (cost=45758 rows=65272) (actual time=0.0917..236 rows=71134 loops=1)
                                -> Nested loop inner join  (cost=45758 rows=65272) (actual time=0.0828..170 rows=71134 loops=1)
                                    -> Nested loop inner join  (cost=22913 rows=65272) (actual time=0.074..89.3 rows=71134 loops=1)
                                        -> Table scan on s  (cost=67.5 rows=615) (actual time=0.0128..0.841 rows=641 loops=1)
                                        -> Index lookup on sd using school_id (school_id=s.School_ID)  (cost=26.6 rows=106) (actual time=0.0814..0.131 rows=111 loops=641)
                                    -> Single-row index lookup on h using PRIMARY (index=sd.housing_id)  (cost=0.25 rows=1) (actual time=944e-6..973e-6 rows=1 loops=71134)
    -> Covering index lookup on max_scores using <auto_key0> (zip_code=sc.zip_code, max_score=sc.total_score)  (cost=0.25..2.5 rows=10) (actual time=949e-6..980e-6 rows=0.221 loops=263)
        -> Materialize  (cost=0..0 rows=0) (actual time=0.137..0.137 rows=58 loops=1)
            -> Table scan on <temporary>  (actual time=0.102..0.107 rows=58 loops=1)
                -> Aggregate using temporary table  (actual time=0.101..0.101 rows=58 loops=1)
                    -> Table scan on ScoredCommunities  (cost=2.5..2.5 rows=0) (actual time=0.0019..0.031 rows=263 loops=1)
                        -> Materialize CTE ScoredCommunities if needed (query plan printed elsewhere)  (cost=0..0 rows=0) (never executed)
 |
1 row in set (0.32 sec)
```

Unset

```
CREATE INDEX dist ON school_district(`distance`);
```



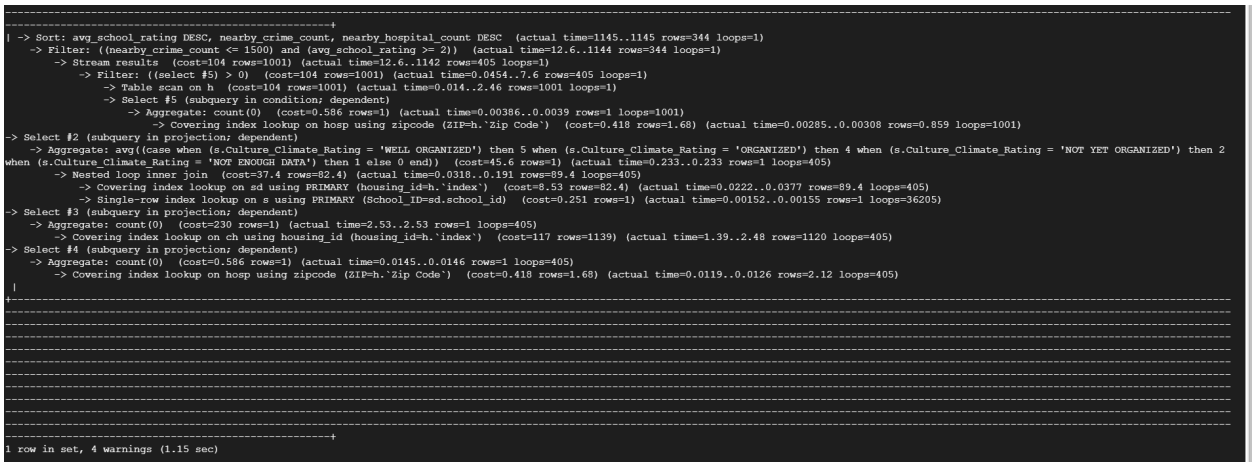
```
-> Nested loop inner join  (cost=163188 rows=0) (actual time=306..306 rows=58 loops=1)
    -> Sort: sc.total_score DESC, sc.avg_distance  (cost=2.6..2.6 rows=0) (actual time=306..306 rows=263 loops=1)
        -> Filter: ((sc.zip_code is not null) and (sc.total_score is not null))  (cost=2.5..2.5 rows=0) (actual time=306..306 rows=263 loops=1)
            -> Table scan on sc  (cost=2.5..2.5 rows=0) (actual time=306..306 rows=263 loops=1)
                -> Materialize CTE ScoredCommunities if needed  (cost=0..0 rows=0) (actual time=306..306 rows=263 loops=1)
                    -> Group aggregate: sum(tmp_field), sum(climate_score), avg(school_district.distance), count(distinct schools.school_id)  (actual time=276..305 rows=263 loops=1)
                        -> Sort: RatedSchools.community_name, RatedSchools.zip_code  (actual time=276..283 rows=71134 loops=1)
                            -> Stream results  (cost=45758 rows=65272) (actual time=0.113..230 rows=71134 loops=1)
                                -> Nested loop inner join  (cost=45758 rows=65272) (actual time=0.104..167 rows=71134 loops=1)
                                    -> Nested loop inner join  (cost=22913 rows=65272) (actual time=0.0967..87.9 rows=71134 loops=1)
                                        -> Table scan on s  (cost=67.5 rows=615) (actual time=0.026..0.982 rows=641 loops=1)
                                        -> Index lookup on sd using school_id (school_id=s.School_ID)  (cost=26.6 rows=106) (actual time=0.0815..0.129 rows=111 loops=641)
                                    -> Single-row index lookup on h using PRIMARY (index=sd.housing_id)  (cost=0.25 rows=1) (actual time=921e-6..950e-6 rows=1 loops=71134)
    -> Covering index lookup on max_scores using <auto_key0> (zip_code=sc.zip_code, max_score=sc.total_score)  (cost=0.25..2.5 rows=10) (actual time=951e-6..983e-6 rows=0.221 loops=263)
        -> Materialize  (cost=0..0 rows=0) (actual time=0.137..0.137 rows=58 loops=1)
            -> Table scan on <temporary>  (actual time=0.101..0.107 rows=58 loops=1)
                -> Aggregate using temporary table  (actual time=0.101..0.101 rows=58 loops=1)
                    -> Table scan on ScoredCommunities  (cost=2.5..2.5 rows=0) (actual time=0.00218..0.0309 rows=263 loops=1)
                        -> Materialize CTE ScoredCommunities if needed (query plan printed elsewhere)  (cost=0..0 rows=0) (never executed)
 |
1 row in set (0.31 sec)
```

The original query had a cost of 163188 before indexing. We decided to use various combinations of `Zip Code` from the housing table, `Community Area Name` from the housing table, and `distance` from the school_distance table. All of these attributes were used in either HAVING, GROUP BY, and JOIN clauses. For our first indexing design, we just added an index on `Zip Code` from the housing table. The cost of the

query after implementing this design stayed at 163188. We decided to keep the previous index and add indexing on the `Community Area Name` as a part of our second design. The cost of the query after this was 163188. For our third design, we decided to keep the previous two indexes and add an additional index to `distance`. The cost of the query after this was still 163188. I think this change in indexing did not create much of a difference in the overall cost, as these attributes were not a part of the joins or filtering in a meaningful way. In the end, we decided to stick with the original query to reduce the added overhead of maintaining the indexes.

QUERY 4:

ORIGINAL



Unset

```
CREATE INDEX zipcode ON housing (`Zip Code`);
```

```
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
| -> Sort: avg_school_rating DESC, nearby_crime_count, nearby_hospital_count DESC  (actual time=144..144 rows=344 loops=1)
    -> Filter: ((nearby_crime_count <= 1500) and (avg_school_rating >= 2))  (actual time=0.424..143 rows=344 loops=1)
        -> Stream results  (cost=104 rows=1001) (actual time=0.421..143 rows=405 loops=1)
            -> Filter: ((select #5) > 0)  (cost=104 rows=1001) (actual time=0.0414..2.75 rows=405 loops=1)
                -> Table scan on h  (cost=104 rows=1001) (actual time=0.0107..0.678 rows=1001 loops=1)
                -> Select #5 (subquery in condition; dependent)
                    -> Aggregate: count(0)  (cost=0.586 rows=1) (actual time=0.00158..0.00161 rows=1 loops=1001)
                        -> Covering index lookup on hosp using zipcode (ZIP=h.`Zip Code`)  (cost=0.418 rows=1.68) (actual time=0.00113..0.00131 rows=0.859 loops=1001)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: avg((case when (s.Culture_Climate_Rating = 'WELL ORGANIZED') then 5 when (s.Culture_Climate_Rating = 'ORGANIZED') then 4 when (s.Culture_Climate_Rating = 'NOT YET ORGANIZED') then 2
when (s.Culture_Climate_Rating = 'NOT ENOUGH DATA') then 1 else 0 end))  (cost=45.6 rows=1) (actual time=0.136..0.136 rows=1 loops=405)
        -> Nested loop inner join  (cost=37.4 rows=82.4) (actual time=0.0119..0.111 rows=89.4 loops=405)
            -> Covering index lookup on sd using PRIMARY (housing_id=h.`index`)  (cost=8.53 rows=82.4) (actual time=0.0103..0.0218 rows=89.4 loops=405)
            -> Single-row index lookup on s using PRIMARY (School_ID=sd.school_id)  (cost=0.251 rows=1) (actual time=812e-6..841e-6 rows=1 loops=36205)
-> Select #3 (subquery in projection; dependent)
    -> Aggregate: count(0)  (cost=229 rows=1) (actual time=0.206..0.206 rows=1 loops=405)
        -> Covering index lookup on ch using housing_id (housing_id=h.`index`)  (cost=115 rows=1139) (actual time=0.0125..0.167 rows=1120 loops=405)
-> Select #4 (subquery in projection; dependent)
    -> Aggregate: count(0)  (cost=0.586 rows=1) (actual time=0.00233..0.00236 rows=1 loops=405)
        -> Covering index lookup on hosp using zipcode (ZIP=h.`Zip Code`)  (cost=0.418 rows=1.68) (actual time=0.00151..0.0019 rows=2.12 loops=405)
 |
+----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
1 row in set, 4 warnings (0.15 sec)
```

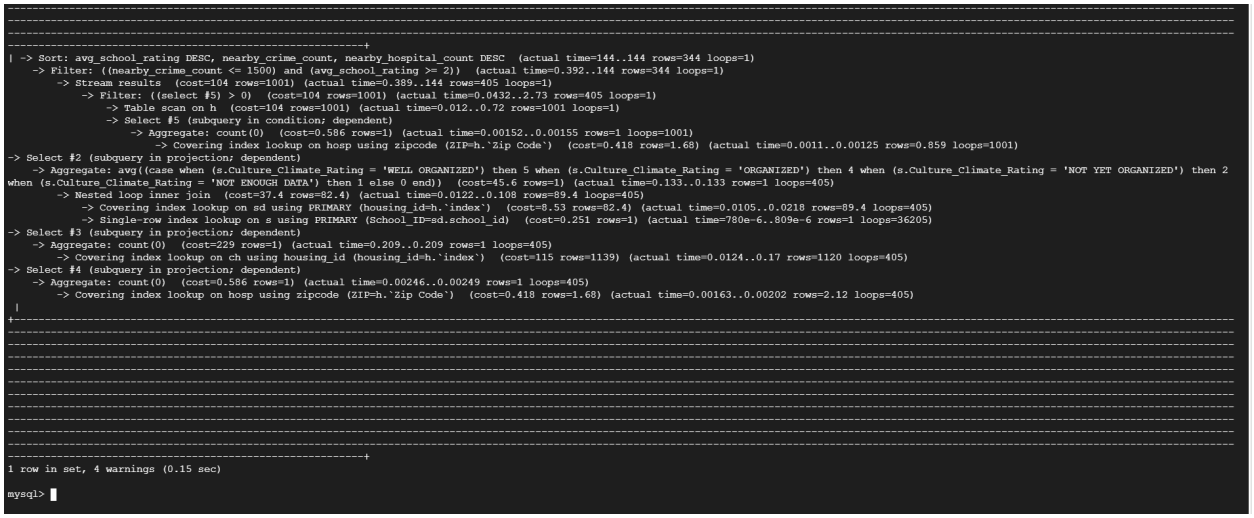
Unset

```
CREATE INDEX ccr ON schools (`culture_climate_rating`(255));
```

```
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
| -> Sort: avg_school_rating DESC, nearby_crime_count, nearby_hospital_count DESC  (actual time=141..141 rows=344 loops=1)
    -> Filter: ((nearby_crime_count <= 1500) and (avg_school_rating >= 2))  (actual time=0.455..141 rows=344 loops=1)
        -> Stream results  (cost=104 rows=1001) (actual time=0.452..141 rows=405 loops=1)
            -> Filter: ((select #5) > 0)  (cost=104 rows=1001) (actual time=0.0406..2.75 rows=405 loops=1)
                -> Table scan on h  (cost=104 rows=1001) (actual time=0.0113..0.7 rows=1001 loops=1)
                -> Select #5 (subquery in condition; dependent)
                    -> Aggregate: count(0)  (cost=0.586 rows=1) (actual time=0.00156..0.00159 rows=1 loops=1001)
                        -> Covering index lookup on hosp using zipcode (ZIP=h.`Zip Code`)  (cost=0.418 rows=1.68) (actual time=0.00112..0.00128 rows=0.859 loops=1001)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: avg((case when (s.Culture_Climate_Rating = 'WELL ORGANIZED') then 5 when (s.Culture_Climate_Rating = 'ORGANIZED') then 4 when (s.Culture_Climate_Rating = 'NOT YET ORGANIZED') then 2
when (s.Culture_Climate_Rating = 'NOT ENOUGH DATA') then 1 else 0 end))  (cost=45.6 rows=1) (actual time=0.134..0.134 rows=1 loops=405)
        -> Nested loop inner join  (cost=37.4 rows=82.4) (actual time=0.0118..0.11 rows=89.4 loops=405)
            -> Covering index lookup on sd using PRIMARY (housing_id=h.`index`)  (cost=8.53 rows=82.4) (actual time=0.0101..0.0211 rows=89.4 loops=405)
            -> Single-row index lookup on s using PRIMARY (School_ID=sd.school_id)  (cost=0.251 rows=1) (actual time=809e-6..838e-6 rows=1 loops=36205)
-> Select #3 (subquery in projection; dependent)
    -> Aggregate: count(0)  (cost=229 rows=1) (actual time=0.202..0.202 rows=1 loops=405)
        -> Covering index lookup on ch using housing_id (housing_id=h.`index`)  (cost=115 rows=1139) (actual time=0.0122..0.162 rows=1120 loops=405)
-> Select #4 (subquery in projection; dependent)
    -> Aggregate: count(0)  (cost=0.586 rows=1) (actual time=0.0024..0.00243 rows=1 loops=405)
        -> Covering index lookup on hosp using zipcode (ZIP=h.`Zip Code`)  (cost=0.418 rows=1.68) (actual time=0.00158..0.00198 rows=2.12 loops=405)
 |
+----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
1 row in set, 4 warnings (0.15 sec)

mysql>
```

Unset

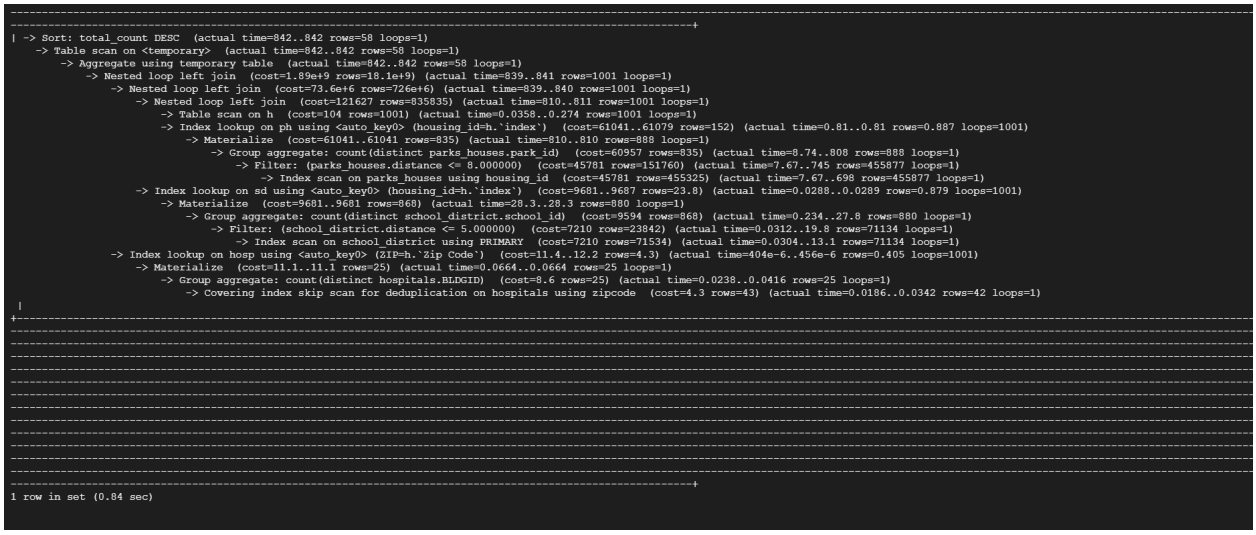
```
CREATE INDEX pn ON housing (`Property Name`(255));
```



The original query had a cost of 104 before indexing. We decided to use various combinations of `Zip Code` from the housing table, `culture_climate_rating` from the schools table, and `Property Name` from the housing table. All of these attributes were used in either HAVING, GROUP BY, and JOIN clauses. For our first indexing design, we just added an index on `Zip Code` from the housing table. The cost of the query after implementing this design stayed at 104. We decided to add indexing on the `culture_climate_rating` as a part of our second design. The cost of the query after this was 104. For our third design, we decided to keep the previous two indexes and add an additional to `Property Name`. The cost of the query after this was still 104. I think this change in indexing did not create much of a difference in the overall cost, as these attributes were not a part of the joins or filtering in a meaningful way. In the end,we decided to stick with the original query to reduce the added overhead of maintaining the indexes.

QUERY 5:

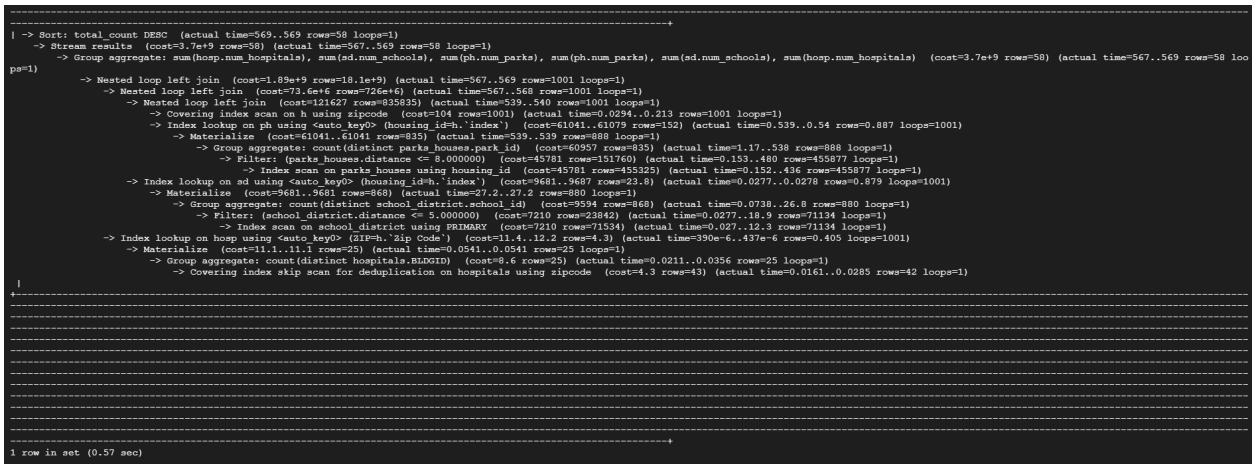
ORIGINAL



```
| -> Sort: total_count DESC  (actual time=842..842 rows=58 loops=1)
    -> Table scan on <temporary>  (actual time=842..842 rows=58 loops=1)
        -> Aggregate using temporary table  (actual time=842..842 rows=58 loops=1)
            -> Nested loop left join  (cost=1.89e+9 rows=18.1e+9) (actual time=839..841 rows=1001 loops=1)
                -> Nested loop left join  (cost=73.6e+6 rows=726e+6) (actual time=839..840 rows=1001 loops=1)
                    -> Nested loop left join  (cost=121627 rows=835835) (actual time=810..811 rows=1001 loops=1)
                        -> Table scan on h  (cost=104 rows=1001) (actual time=0.0358..0.274 rows=1001 loops=1)
                        -> Index lookup on ph using <auto_key0> (housing_id=h.`index`)  (cost=61041..61079 rows=152) (actual time=0.81..0.81 rows=0.887 loops=1001)
                            -> Materialize  (cost=61041..61041 rows=835) (actual time=810..810 rows=888 loops=1)
                                -> Group aggregate: count(distinct parks_houses.park_id)  (cost=60957 rows=835) (actual time=8.74..808 rows=888 loops=1)
                                    -> Filter: (parks_houses.distance <= 8.000000)  (cost=45781 rows=151760) (actual time=7.67..745 rows=455877 loops=1)
                                        -> Index scan on parks_houses using housing_id  (cost=45781 rows=455325) (actual time=7.67..698 rows=455877 loops=1)
                    -> Index lookup on sd using <auto_key0> (housing_id=h.`index`)  (cost=9681..9687 rows=23.8) (actual time=0.0288..0.0289 rows=0.879 loops=1001)
                        -> Materialize  (cost=9681..9681 rows=868) (actual time=28.3..28.3 rows=880 loops=1)
                            -> Group aggregate: count(distinct school_district.school_id)  (cost=9594 rows=868) (actual time=0.234..27.8 rows=880 loops=1)
                                -> Filter: (school_district.distance <= 5.000000)  (cost=7210 rows=23842) (actual time=0.0312..19.8 rows=71134 loops=1)
                                    -> Index scan on school_district using PRIMARY  (cost=7210 rows=71534) (actual time=0.0304..13.1 rows=71134 loops=1)
                -> Index lookup on hosp using <auto_key0> (ZIP=h.`Zip Code`)  (cost=11.4..12.2 rows=4.3) (actual time=404e-6..456e-6 rows=0.405 loops=1001)
                    -> Materialize  (cost=11.1..11.1 rows=25) (actual time=0.0664..0.0664 rows=25 loops=1)
                        -> Group aggregate: count(distinct hospitals.BLDGID)  (cost=8.6 rows=25) (actual time=0.0238..0.0416 rows=25 loops=1)
                            -> Covering index skip scan for deduplication on hospitals using zipcode  (cost=4.3 rows=43) (actual time=0.0186..0.0342 rows=42 loops=1)
 |
1 row in set (0.84 sec)
```

Unset

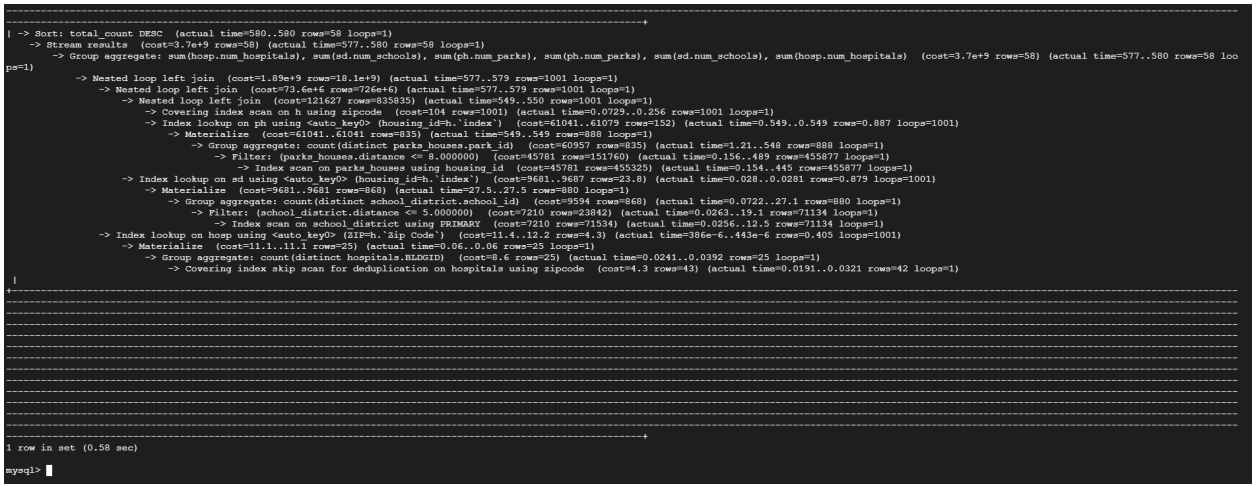
CREATE INDEX zipcode ON housing (`Zip Code`);



```
| -> Sort: total_count DESC  (actual time=569..569 rows=58 loops=1)
    -> Stream results  (cost=3.7e+9 rows=58) (actual time=567..569 rows=58 loops=1)
        -> Group aggregate: sum(hosp.num_hospitals), sum(sd.num_schools), sum(ph.num_parks), sum(ph.num_parks), sum(sd.num_schools), sum(hosp.num_hospitals)  (cost=3.7e+9 rows=58) (actual time=567..569 rows=58 loo
ps=1)
            -> Nested loop left join  (cost=1.89e+9 rows=18.1e+9) (actual time=567..569 rows=1001 loops=1)
                -> Nested loop left join  (cost=73.6e+6 rows=726e+6) (actual time=567..568 rows=1001 loops=1)
                    -> Nested loop left join  (cost=121627 rows=835835) (actual time=539..540 rows=1001 loops=1)
                        -> Covering index scan on h using zipcode  (cost=104 rows=1001) (actual time=0.0294..0.213 rows=1001 loops=1)
                        -> Index lookup on ph using <auto_key0> (housing_id=h.`index`)  (cost=61041..61079 rows=152) (actual time=0.539..0.54 rows=0.887 loops=1001)
                            -> Materialize  (cost=61041..61041 rows=835) (actual time=539..539 rows=888 loops=1)
                                -> Group aggregate: count(distinct parks_houses.park_id)  (cost=60957 rows=835) (actual time=1.17..538 rows=888 loops=1)
                                    -> Filter: (parks_houses.distance <= 8.000000)  (cost=45781 rows=151760) (actual time=0.153..480 rows=455877 loops=1)
                                        -> Index scan on parks_houses using housing_id  (cost=45781 rows=455325) (actual time=0.152..436 rows=455877 loops=1)
                    -> Index lookup on sd using <auto_key0> (housing_id=h.`index`)  (cost=9681..9687 rows=23.8) (actual time=0.0277..0.0278 rows=0.879 loops=1001)
                        -> Materialize  (cost=9681..9681 rows=868) (actual time=27.2..27.2 rows=880 loops=1)
                            -> Group aggregate: count(distinct school_district.school_id)  (cost=9594 rows=868) (actual time=0.0738..26.8 rows=880 loops=1)
                                -> Filter: (school_district.distance <= 5.000000)  (cost=7210 rows=23842) (actual time=0.0277..18.9 rows=71134 loops=1)
                                    -> Index scan on school_district using PRIMARY  (cost=7210 rows=71534) (actual time=0.027..12.3 rows=71134 loops=1)
                -> Index lookup on hosp using <auto_key0> (ZIP=h.`Zip Code`)  (cost=11.4..12.2 rows=4.3) (actual time=390e-6..437e-6 rows=0.405 loops=1001)
                    -> Materialize  (cost=11.1..11.1 rows=25) (actual time=0.0541..0.0541 rows=25 loops=1)
                        -> Group aggregate: count(distinct hospitals.BLDGID)  (cost=8.6 rows=25) (actual time=0.0211..0.0356 rows=25 loops=1)
                            -> Covering index skip scan for deduplication on hospitals using zipcode  (cost=4.3 rows=43) (actual time=0.0161..0.0285 rows=42 loops=1)
 |
1 row in set (0.57 sec)
```

Unset

CREATE INDEX zipcode2 ON hospitals (`ZIP`);



The original query had a cost of 1.89e+9 before indexing. We decided to use various combinations of `Zip Code` from the housing table and `ZIP` from the hospitals table. All of these attributes were used in either HAVING, GROUP BY, and JOIN clauses. For our first indexing design, we just added an index on `Zip Code` from the housing table. The cost of the query after implementing this design stayed at 1.89e+9. We decided to add indexing on only the `ZIP` column as a part of our second design. The cost of the query after this was 1.89e+9. For our third design, we decided to keep the previous index and readded the `Zip Code` index. The cost of the query after this was still 1.89e+9. We weren't really sure why the cost remained so high even after adding indexing. We were unable to add any more indexes, as every other column used was a primary key. One possibility could be that since our query has a lot of joins using primary keys, our indexing didn't really affect the overall cost. In the end, we decided to stick with the original query to reduce the added overhead of maintaining the indexes.