Database Implementation

DDL Commands

```
create table Location(
      block int PRIMARY KEY,
      has_police_station bool
);

create table Apartment (
      address varchar(255) PRIMARY KEY,
safestay_score float,
      latitude float,
      longitude float,
      block int,
FOREIGN KEY (block) REFERENCES Location(block)
);

create table Streetlight (
      streetlight_id int PRIMARY KEY,
pole_material varchar(255),
      wattage int,
      height int,
color varchar(255),
      latitude float,
      longitude float,
block int,
      FOREIGN KEY (block) REFERENCES Location(block)
);

create table Pedestrian_Crash (
      crash_id int PRIMARY KEY,
      crash_severity varchar(255),
      traffic_control varchar(255),
year INT,
road_surface varchar(255),
latitude float,
```

```
longitude float,
block int,
FOREIGN KEY (block) REFERENCES Location(block)
);

create table User (
      username varchar(255) PRIMARY KEY,
      password varchar(255),
      first_name varchar(255),
      last_name varchar(255)
);

create table Rating (
      username varchar(255),
      address varchar(255),
      rating int,
      PRIMARY KEY(username, address),
      FOREIGN KEY (username) REFERENCES User(username),
      FOREIGN KEY (address) REFERENCES Apartment(address)
);
```

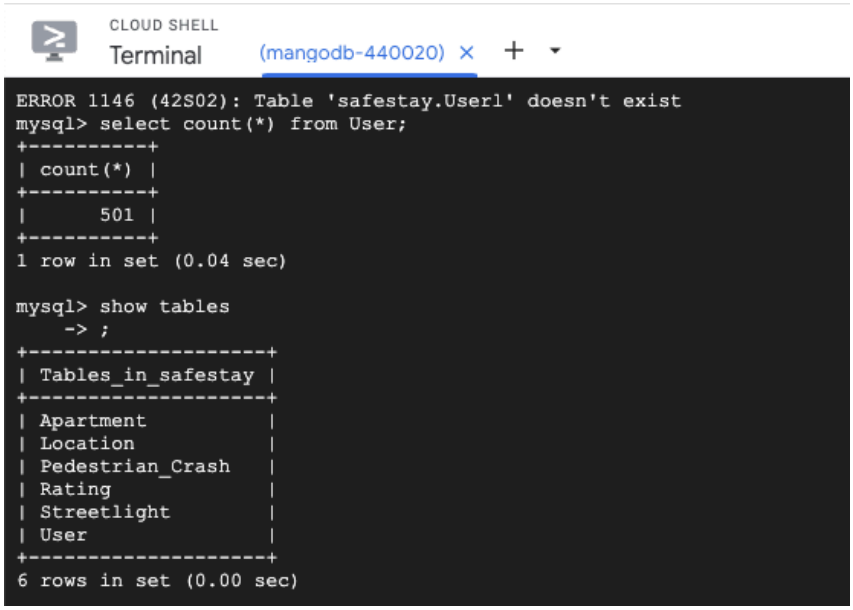
Count Queries

```
mysql> select count(*) from Apartment;
+----------+
| count(*) |
+----------+
|     2585 |
+----------+
1 row in set (0.06 sec)
```

```
mysql> select count(*) from Streetlight;
+----------+
| count(*) |
+----------+
|     4557 |
+----------+
1 row in set (0.10 sec)
```

```
mysql> select count(*) from Rating;
+----------+
| count(*) |
+----------+
|     5001 |
+----------+
1 row in set (0.10 sec)
```

Implementation in GCP



Advanced Queries

Query 1

Address and community score for apartments where there are no streetlights with poles not made out of wood

```
SELECT A.address, AVG(R.rating) AS avg_community_score
FROM Apartment A
JOIN Rating R ON A.address = R.address
WHERE A.block NOT IN (
    SELECT S.block
    FROM Streetlight S
    WHERE S.pole_material NOT LIKE '%Wood%'
)
GROUP BY A.address
ORDER BY AVG(R.rating) ASC
LIMIT 15;
```

```
mysql> SELECT A.address, AVG(R.rating) AS avg_community_score
    -> FROM Apartment A
    -> JOIN Rating R ON A.address = R.address
    -> WHERE A.block NOT IN (
    ->     SELECT S.block
    ->     FROM Streetlight S
    ->     WHERE S.pole_material NOT LIKE '%Wood%'
    -> )
    -> GROUP BY A.address
    -> ORDER BY AVG(R.rating) ASC
    -> LIMIT 15;
+-------------------+---------------------+
| address           | avg_community_score |
+-------------------+---------------------+
| 3211 W Kirby Ave  |              0.0000 |
| 4608 Nicklaus Dr  |              0.0000 |
| 3737 Thornhill Dr |              0.0000 |
| 3846 Balmoral Dr  |              0.0000 |
| 3860 Thornhill Dr |              0.0000 |
| 3770 Thornhill Dr |              0.0000 |
| 3716 Thornhill Dr |              0.0000 |
| 3762 Thornhill Dr |              0.0000 |
| 35 Colony West Dr |              0.5000 |
| 2902 Watterson Ct |              1.0000 |
| 1602 Holmstrom Dr |              1.0000 |
| 3766 Thornhill Dr |              1.0000 |
| 1819 Valley Rd    |              1.0000 |
| 3702 Thornhill Dr |              1.0000 |
| 4606 Nicklaus Dr  |              1.0000 |
+-------------------+---------------------+
15 rows in set (0.03 sec)
```

Query 2

Apartments With above average SafeStay score, police station in same block, and above average community score

```
(SELECT A.address
FROM Apartment A
JOIN Location L ON A.block = L.block
WHERE A.safestay_score > (
    SELECT AVG(safestay_score)
    FROM Apartment
)
AND L.has_police_station = TRUE)

INTERSECT

(SELECT A.address
FROM Apartment A
```

```
JOIN Rating R ON A.address = R.address
GROUP BY A.address
HAVING AVG(R.rating) > (
    SELECT AVG(rating)
    FROM Rating
))
LIMIT 15;
```

```
mysql> (SELECT A.address
    -> FROM Apartment A
    -> JOIN Location L ON A.block = L.block
    -> WHERE A.safestay_score > (
    ->     SELECT AVG(safestay_score)
    ->     FROM Apartment
    -> )
    -> AND L.has_police_station = TRUE)
    -> INTERSECT
    -> (SELECT A.address
    -> FROM Apartment A
    -> JOIN Rating R ON A.address = R.address
    -> GROUP BY A.address
    -> HAVING AVG(R.rating) > (
    ->     SELECT AVG(rating)
    ->     FROM Rating
    -> ))
    -> LIMIT 15;
+-----------------------------+
| address                     |
+-----------------------------+
| 1601-1603 Congressional Way |
| 4510 Nicklaus Dr            |
| 2408 Fields South Drive     |
| 2506 Fields South Drive     |
| 2508 Fields South Drive     |
| 2510 Fields South Drive     |
| 2512 Fields South Drive     |
| 2514 Fields South Drive     |
| 1903 Melrose Dr             |
| 1911 Melrose Dr             |
| 1914 Melrose Dr             |
| 1929 Melrose Dr             |
| 2 Colony West Dr            |
| 2102 Melrose Dr             |
| 2104 Melrose Dr             |
+-----------------------------+
```

Query 3

Find the Average Crash Severity Near Apartments With No Police Stations

```
ALTER TABLE Pedestrian_Crash ADD COLUMN crash_severity_number INT;
UPDATE Pedestrian_Crash
```

```
SET crash_severity_number = CASE
    WHEN crash_severity = 'No Injuries' THEN 1
    WHEN crash_severity = 'Property Damage' THEN 2
    WHEN crash_severity = 'C-Injury' THEN 3
    WHEN crash_severity = 'C Injury Crash' THEN 3
    WHEN crash_severity = 'B-Injury' THEN 4
    WHEN crash_severity = 'B Injury Crash' THEN 4
    WHEN crash_severity = 'A-Injury' THEN 5
    WHEN crash_severity = 'A Injury Crash' THEN 5
    WHEN crash_severity = 'Fatal' THEN 6
    WHEN crash_severity = 'Fatal Crash' THEN 6
    ELSE NULL
END;
```

```
SELECT address, AVG(crash_severity_number) as average_crash_severity_rating
FROM Pedestrian_Crash JOIN Apartment USING(Block) JOIN Location
USING(Block)
WHERE has_police_station = 'FALSE'
GROUP BY address
```

```
mysql> SELECT address, AVG(crash_severity_number) as average_crash_severity_rating
    -> FROM Pedestrian_Crash JOIN Apartment USING(Block) JOIN Location USING(Block)
    -> WHERE has_police_station = 'FALSE'
    -> GROUP BY address Limit 15;
+--------------------+--------------------------------+
| address            | average_crash_severity_rating |
+--------------------+--------------------------------+
| 1 Colony West Dr   |                         4.6667 |
| 1 Main St          |                         4.5000 |
| 10 1/2 Main St     |                         3.9787 |
| 100 Kenwood Rd     |                         4.5294 |
| 1001 S Wright St   |                         4.0268 |
| 1002 S Second St   |                         4.0268 |
| 1003 N Randolph St |                         4.5000 |
| 1004 S First St    |                         4.0268 |
| 1004 S Second St   |                         4.0268 |
| 1005 S First St    |                         4.0268 |
| 1005 S Second St   |                         4.0268 |
| 1005 S Sixth St    |                         4.0268 |
| 1005 S Wright St   |                         4.0268 |
| 1006 S Third St    |                         4.0268 |
| 1007 N Randolph St |                         4.5000 |
+--------------------+--------------------------------+
15 rows in set, 1 warning (0.01 sec)
```

Query 4

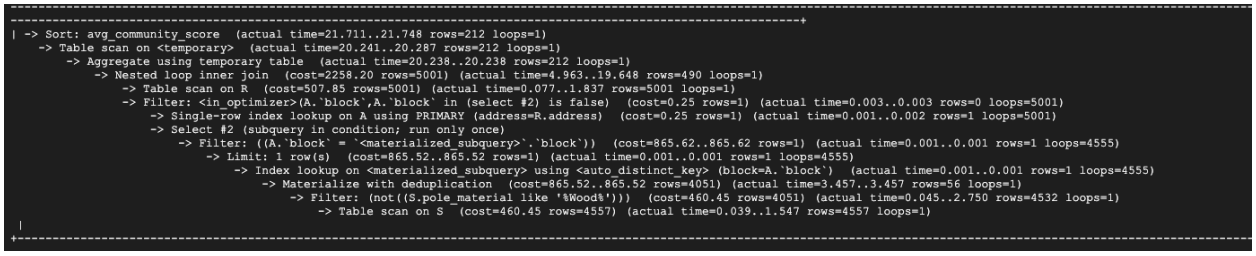
Identify Locations Where Streetlight Wattage Is Below Average and Crashes Are High

```
WITH AvgWattagePerBlock AS ( SELECT block, AVG(wattage) AS avg_wattage FROM Streetlight GROUP BY block ), CrashesPerBlock AS ( SELECT block, COUNT(*) AS num_crashes FROM Pedestrian_Crash GROUP BY block ), AverageValues AS ( SELECT (SELECT AVG(wattage) FROM Streetlight) AS overall_avg_wattage, (SELECT AVG(num_crashes) FROM CrashesPerBlock) AS overall_avg_crashes ) SELECT l.block FROM Location l JOIN AvgWattagePerBlock w USING(block) JOIN CrashesPerBlock c USING(block) CROSS JOIN AverageValues a WHERE w.avg_wattage < a.overall_avg_wattage AND c.num_crashes > a.overall_avg_crashes;
```

```
mysql> WITH AvgWattagePerBlock AS (
    ->     SELECT block, AVG(wattage) AS avg_wattage
    ->     FROM Streetlight
    ->     GROUP BY block
    -> ),
    -> CrashesPerBlock AS (
    ->     SELECT block, COUNT(*) AS num_crashes
    ->     FROM Pedestrian_Crash
    ->     GROUP BY block
    -> ),
    -> AverageValues AS (
    ->     SELECT
    ->         (SELECT AVG(wattage) FROM Streetlight) AS overall_avg_wattage,
    ->         (SELECT AVG(num_crashes) FROM CrashesPerBlock) AS overall_avg_crashes
    -> )
    -> SELECT l.block
    -> FROM Location l
    -> JOIN AvgWattagePerBlock w USING(block)
    -> JOIN CrashesPerBlock c USING(block)
    -> CROSS JOIN AverageValues a
    -> WHERE w.avg_wattage < a.overall_avg_wattage
    ->   AND c.num_crashes > a.overall_avg_crashes;
+-------+
| block |
+-------+
|    44 |
|    45 |
|    47 |
|    56 |
|    57 |
+-------+
5 rows in set (0.02 sec)
```

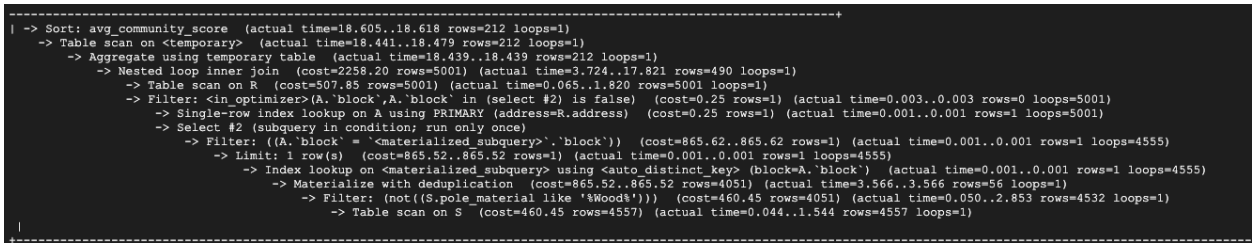
Indexing

Query 1



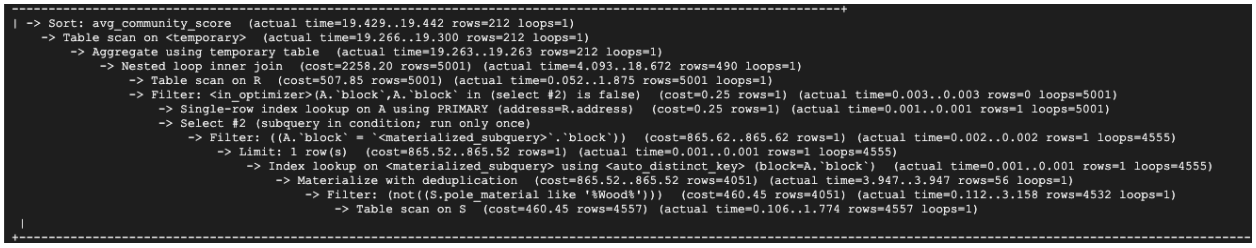
```
| -> Sort: avg_community_score  (actual time=21.711..21.748 rows=212 loops=1)
    -> Table scan on <temporary>  (actual time=20.241..20.287 rows=212 loops=1)
        -> Aggregate using temporary table  (actual time=20.238..20.238 rows=212 loops=1)
            -> Nested loop inner join  (cost=2258.20 rows=5001) (actual time=4.963..19.648 rows=490 loops=1)
                -> Table scan on R  (cost=507.85 rows=5001) (actual time=0.077..1.837 rows=5001 loops=1)
                -> Filter: <in_optimizer>(A.`block`,A.`block` in (select #2) is false)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=0 loops=5001)
                    -> Single-row index lookup on A using PRIMARY (address=R.address)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=5001)
                    -> Select #2 (subquery in condition; run only once)
                        -> Filter: ((A.`block` = `<materialized_subquery>`.`block`))  (cost=865.62..865.62 rows=1) (actual time=0.001..0.001 rows=1 loops=4555)
                            -> Limit: 1 row(s)  (cost=865.52..865.52 rows=1) (actual time=0.001..0.001 rows=1 loops=4555)
                                -> Index lookup on <materialized_subquery> using <auto_distinct_key> (block=A.`block`)  (actual time=0.001..0.001 rows=1 loops=4555)
                                    -> Materialize with deduplication  (cost=865.52..865.52 rows=4051) (actual time=3.457..3.457 rows=56 loops=1)
                                        -> Filter: (not((S.pole_material like '%Wood%')))  (cost=460.45 rows=4051) (actual time=0.045..2.750 rows=4532 loops=1)
                                            -> Table scan on S  (cost=460.45 rows=4557) (actual time=0.039..1.547 rows=4557 loops=1)
```

create index apartment_block on Apartment(block);



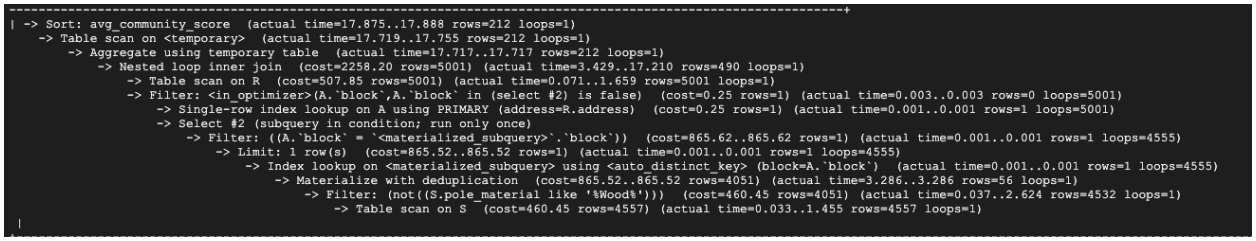
```
| -> Sort: avg_community_score  (actual time=18.605..18.618 rows=212 loops=1)
    -> Table scan on <temporary>  (actual time=18.441..18.479 rows=212 loops=1)
        -> Aggregate using temporary table  (actual time=18.439..18.439 rows=212 loops=1)
            -> Nested loop inner join  (cost=2258.20 rows=5001) (actual time=3.724..17.821 rows=490 loops=1)
                -> Table scan on R  (cost=507.85 rows=5001) (actual time=0.065..1.820 rows=5001 loops=1)
                -> Filter: <in_optimizer>(A.`block`,A.`block` in (select #2) is false)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=0 loops=5001)
                    -> Single-row index lookup on A using PRIMARY (address=R.address)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5001)
                    -> Select #2 (subquery in condition; run only once)
                        -> Filter: ((A.`block` = `<materialized_subquery>`.`block`))  (cost=865.62..865.62 rows=1) (actual time=0.001..0.001 rows=1 loops=4555)
                            -> Limit: 1 row(s)  (cost=865.52..865.52 rows=1) (actual time=0.001..0.001 rows=1 loops=4555)
                                -> Index lookup on <materialized_subquery> using <auto_distinct_key> (block=A.`block`)  (actual time=0.001..0.001 rows=1 loops=4555)
                                    -> Materialize with deduplication  (cost=865.52..865.52 rows=4051) (actual time=3.566..3.566 rows=56 loops=1)
                                        -> Filter: (not((S.pole_material like '%Wood%')))  (cost=460.45 rows=4051) (actual time=0.050..2.853 rows=4532 loops=1)
                                            -> Table scan on S  (cost=460.45 rows=4557) (actual time=0.044..1.544 rows=4557 loops=1)
```

create index streetlight_block on Streetlight(block);



```
| -> Sort: avg_community_score  (actual time=19.429..19.442 rows=212 loops=1)
    -> Table scan on <temporary>  (actual time=19.266..19.300 rows=212 loops=1)
        -> Aggregate using temporary table  (actual time=19.263..19.263 rows=212 loops=1)
            -> Nested loop inner join  (cost=2258.20 rows=5001) (actual time=4.093..18.672 rows=490 loops=1)
                -> Table scan on R  (cost=507.85 rows=5001) (actual time=0.052..1.875 rows=5001 loops=1)
                -> Filter: <in_optimizer>(A.`block`,A.`block` in (select #2) is false)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=0 loops=5001)
                    -> Single-row index lookup on A using PRIMARY (address=R.address)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5001)
                    -> Select #2 (subquery in condition; run only once)
                        -> Filter: ((A.`block` = `<materialized_subquery>`.`block`))  (cost=865.62..865.62 rows=1) (actual time=0.002..0.002 rows=1 loops=4555)
                            -> Limit: 1 row(s)  (cost=865.52..865.52 rows=1) (actual time=0.001..0.001 rows=1 loops=4555)
                                -> Index lookup on <materialized_subquery> using <auto_distinct_key> (block=A.`block`)  (actual time=0.001..0.001 rows=1 loops=4555)
                                    -> Materialize with deduplication  (cost=865.52..865.52 rows=4051) (actual time=3.947..3.947 rows=56 loops=1)
                                        -> Filter: (not((S.pole_material like '%Wood%')))  (cost=460.45 rows=4051) (actual time=0.112..3.158 rows=4532 loops=1)
                                            -> Table scan on S  (cost=460.45 rows=4557) (actual time=0.106..1.774 rows=4557 loops=1)
```

create index streetlight_material on Streetlight(pole_material);



```
| -> Sort: avg_community_score  (actual time=17.875..17.888 rows=212 loops=1)
    -> Table scan on <temporary>  (actual time=17.719..17.755 rows=212 loops=1)
        -> Aggregate using temporary table  (actual time=17.717..17.717 rows=212 loops=1)
            -> Nested loop inner join  (cost=2258.20 rows=5001) (actual time=3.429..17.210 rows=490 loops=1)
                -> Table scan on R  (cost=507.85 rows=5001) (actual time=0.071..1.659 rows=5001 loops=1)
                -> Filter: <in_optimizer>(A.`block`,A.`block` in (select #2) is false)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=0 loops=5001)
                    -> Single-row index lookup on A using PRIMARY (address=R.address)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5001)
                    -> Select #2 (subquery in condition; run only once)
                        -> Filter: ((A.`block` = `<materialized_subquery>`.`block`))  (cost=865.62..865.62 rows=1) (actual time=0.001..0.001 rows=1 loops=4555)
                            -> Limit: 1 row(s)  (cost=865.52..865.52 rows=1) (actual time=0.001..0.001 rows=1 loops=4555)
                                -> Index lookup on <materialized_subquery> using <auto_distinct_key> (block=A.`block`)  (actual time=0.001..0.001 rows=1 loops=4555)
                                    -> Materialize with deduplication  (cost=865.52..865.52 rows=4051) (actual time=3.286..3.286 rows=56 loops=1)
                                        -> Filter: (not((S.pole_material like '%Wood%')))  (cost=460.45 rows=4051) (actual time=0.037..2.624 rows=4532 loops=1)
                                            -> Table scan on S  (cost=460.45 rows=4557) (actual time=0.033..1.455 rows=4557 loops=1)
```

For the indexing, we first decided to index on the block in the apartments table. We thought this would be a good idea because we are checking the blocks which contain no streetlights made out of wood. This did not affect the cost at all since the block in apartments is a foreign key and foreign keys are already indexed by the system.

Next we focused on indexing by the blocks in the streetlights table. This did not affect the cost since every block is different and attempting to maximize the ordered efficiency of the database based on blocks is probably the same as randomly searching through them.

Indexing on the pole material in the streetlights table also did not change the cost at all. This was probably because there was no efficient way to index the database and the system probably decided that searching through the entire system was the same as indexing.

Overall I would not recommend indexing for this query because we tried indexing various attributes and nothing reduced the cost of the query. Since this is a relatively simple query, it could be possible that there was no way to optimize the query further.

Query 2

```
| -> Table scan on <intersect temporary>  (cost=190.44..194.87 rows=157) (actual time=22.044..22.081 rows=148 loops=1)
    -> Intersect materialize with deduplication  (cost=190.42..190.42 rows=157) (actual time=22.042..22.042 rows=391 loops=1)
        -> Nested loop inner join  (cost=174.75 rows=157) (actual time=1.703..4.432 rows=391 loops=1)
            -> Filter: (L.has_police_station = true)  (cost=10.25 rows=10) (actual time=0.067..0.096 rows=48 loops=1)
                -> Table scan on L  (cost=10.25 rows=100) (actual time=0.061..0.078 rows=100 loops=1)
            -> Filter: (A.safestay_score > (select #2))  (cost=11.91 rows=16) (actual time=0.062..0.076 rows=8 loops=48)
                -> Index lookup on A using apartment_block (block=L.`block`)  (cost=11.91 rows=47) (actual time=0.043..0.055 rows=18 loops=48)
                -> Select #2 (subquery in condition; run only once)
                    -> Aggregate: avg(Apartment.safestay_score)  (cost=521.50 rows=1) (actual time=0.868..0.869 rows=1 loops=1)
                        -> Table scan on Apartment  (cost=263.00 rows=2585) (actual time=0.041..0.694 rows=2585 loops=1)
        -> Filter: (avg(R.rating) > (select #4))  (actual time=15.502..16.728 rows=1015 loops=1)
            -> Table scan on <temporary>  (actual time=13.862..14.389 rows=2195 loops=1)
                -> Aggregate using temporary table  (actual time=13.860..13.860 rows=2195 loops=1)
                    -> Nested loop inner join  (cost=2258.20 rows=5001) (actual time=0.053..9.272 rows=5001 loops=1)
                        -> Table scan on R  (cost=507.85 rows=5001) (actual time=0.045..1.654 rows=5001 loops=1)
                        -> Single-row covering index lookup on A using PRIMARY (address=R.address)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5001)
            -> Select #4 (subquery in condition; run only once)
                -> Aggregate: avg(Rating.rating)  (cost=1007.95 rows=1) (actual time=1.605..1.605 rows=1 loops=1)
                    -> Table scan on Rating  (cost=507.85 rows=5001) (actual time=0.060..1.141 rows=5001 loops=1)
 |
```

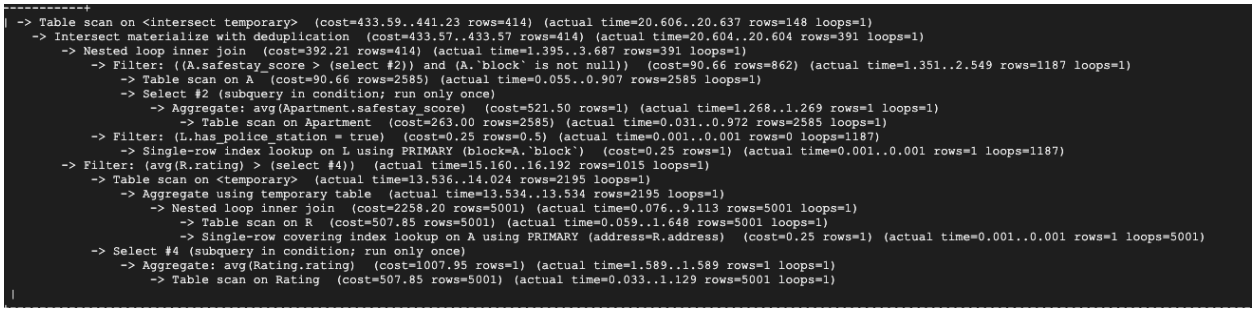
create index apartment_score on Apartment(safestay_score);

```
| -> Table scan on <intersect temporary>  (cost=196.36..201.52 rows=216) (actual time=21.284..21.316 rows=148 loops=1)
    -> Intersect materialize with deduplication  (cost=196.33..196.33 rows=216) (actual time=21.282..21.282 rows=391 loops=1)
        -> Nested loop inner join  (cost=174.75 rows=216) (actual time=0.141..1.998 rows=391 loops=1)
            -> Filter: (L.has_police_station = true)  (cost=10.25 rows=10) (actual time=0.047..0.072 rows=48 loops=1)
                -> Table scan on L  (cost=10.25 rows=100) (actual time=0.044..0.059 rows=100 loops=1)
            -> Filter: (A.safestay_score > (select #2))  (cost=11.97 rows=22) (actual time=0.023..0.039 rows=8 loops=48)
                -> Index lookup on A using apartment_block (block=L.`block`)  (cost=11.97 rows=47) (actual time=0.023..0.037 rows=18 loops=48)
                -> Select #2 (subquery in condition; run only once)
                    -> Aggregate: avg(Apartment.safestay_score)  (cost=521.50 rows=1) (actual time=0.778..0.778 rows=1 loops=1)
                        -> Covering index scan on Apartment using apartment_score  (cost=263.00 rows=2585) (actual time=0.045..0.573 rows=2585 loops=1)
        -> Filter: (avg(R.rating) > (select #4))  (actual time=17.473..18.556 rows=1015 loops=1)
            -> Table scan on <temporary>  (actual time=15.592..16.121 rows=2195 loops=1)
                -> Aggregate using temporary table  (actual time=15.590..15.590 rows=2195 loops=1)
                    -> Nested loop inner join  (cost=2258.20 rows=5001) (actual time=0.046..10.328 rows=5001 loops=1)
                        -> Table scan on R  (cost=507.85 rows=5001) (actual time=0.037..1.931 rows=5001 loops=1)
                        -> Single-row covering index lookup on A using PRIMARY (address=R.address)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5001)
            -> Select #4 (subquery in condition; run only once)
                -> Aggregate: avg(Rating.rating)  (cost=1007.95 rows=1) (actual time=1.842..1.843 rows=1 loops=1)
                    -> Table scan on Rating  (cost=507.85 rows=5001) (actual time=0.038..1.318 rows=5001 loops=1)
 |
```

create index rating_rating on Rating(rating);

```
| -> Table scan on <intersect temporary>  (cost=190.44..194.87 rows=157) (actual time=20.495..20.527 rows=148 loops=1)
    -> Intersect materialize with deduplication  (cost=190.42..190.42 rows=157) (actual time=20.493..20.493 rows=391 loops=1)
        -> Nested loop inner join  (cost=174.75 rows=157) (actual time=1.296..3.088 rows=391 loops=1)
            -> Filter: (L.has_police_station = true)  (cost=10.25 rows=10) (actual time=0.043..0.069 rows=48 loops=1)
                -> Table scan on L  (cost=10.25 rows=100) (actual time=0.040..0.055 rows=100 loops=1)
            -> Filter: (A.safestay_score > (select #2))  (cost=11.91 rows=16) (actual time=0.047..0.062 rows=8 loops=48)
                -> Index lookup on A using apartment_block (block=L.`block`)  (cost=11.91 rows=47) (actual time=0.022..0.035 rows=18 loops=48)
                -> Select #2 (subquery in condition; run only once)
                    -> Aggregate: avg(Apartment.safestay_score)  (cost=521.50 rows=1) (actual time=1.185..1.186 rows=1 loops=1)
                        -> Table scan on Apartment  (cost=263.00 rows=2585) (actual time=0.044..0.981 rows=2585 loops=1)
        -> Filter: (avg(R.rating) > (select #4))  (actual time=15.656..16.663 rows=1015 loops=1)
            -> Table scan on <temporary>  (actual time=14.083..14.556 rows=2195 loops=1)
                -> Aggregate using temporary table  (actual time=14.080..14.080 rows=2195 loops=1)
                    -> Nested loop inner join  (cost=2258.20 rows=5001) (actual time=0.051..9.391 rows=5001 loops=1)
                        -> Covering index scan on R using rating_rating  (cost=507.85 rows=5001) (actual time=0.042..1.560 rows=5001 loops=1)
                        -> Single-row covering index lookup on A using PRIMARY (address=R.address)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5001)
            -> Select #4 (subquery in condition; run only once)
                -> Aggregate: avg(Rating.rating)  (cost=1007.95 rows=1) (actual time=1.536..1.536 rows=1 loops=1)
                    -> Covering index scan on Rating using rating_rating  (cost=507.85 rows=5001) (actual time=0.039..1.071 rows=5001 loops=1)
 |
```

create index location_station on Location(has_police_station);



```
-----------+
| -> Table scan on <intersect temporary>  (cost=433.59..441.23 rows=414) (actual time=20.606..20.637 rows=148 loops=1)
    -> Intersect materialize with deduplication  (cost=433.57..433.57 rows=414) (actual time=20.604..20.604 rows=391 loops=1)
        -> Nested loop inner join  (cost=392.21 rows=414) (actual time=1.395..3.687 rows=391 loops=1)
            -> Filter: ((A.safestay_score > (select #2)) and (A.`block` is not null))  (cost=90.66 rows=862) (actual time=1.351..2.549 rows=1187 loops=1)
                -> Table scan on A  (cost=90.66 rows=2585) (actual time=0.055..0.907 rows=2585 loops=1)
                -> Select #2 (subquery in condition; run only once)
                    -> Aggregate: avg(Apartment.safestay_score)  (cost=521.50 rows=1) (actual time=1.268..1.269 rows=1 loops=1)
                        -> Table scan on Apartment  (cost=263.00 rows=2585) (actual time=0.031..0.972 rows=2585 loops=1)
            -> Filter: (L.has_police_station = true)  (cost=0.25 rows=0.5) (actual time=0.001..0.001 rows=0 loops=1187)
                -> Single-row index lookup on L using PRIMARY (block=A.`block`)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1187)
        -> Filter: (avg(R.rating) > (select #4))  (actual time=15.160..16.192 rows=1015 loops=1)
            -> Table scan on <temporary>  (actual time=13.536..14.024 rows=2195 loops=1)
                -> Aggregate using temporary table  (actual time=13.534..13.534 rows=2195 loops=1)
                    -> Nested loop inner join  (cost=2258.20 rows=5001) (actual time=0.076..9.113 rows=5001 loops=1)
                        -> Table scan on R  (cost=507.85 rows=5001) (actual time=0.059..1.648 rows=5001 loops=1)
                        -> Single-row covering index lookup on A using PRIMARY (address=R.address)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5001)
            -> Select #4 (subquery in condition; run only once)
                -> Aggregate: avg(Rating.rating)  (cost=1007.95 rows=1) (actual time=1.589..1.589 rows=1 loops=1)
                    -> Table scan on Rating  (cost=507.85 rows=5001) (actual time=0.033..1.129 rows=5001 loops=1)
|
```

The slight increase in cost after indexing Apartment(safestay_score) could be because the distribution of safestay_score values isn't very diverse (0-10), so the index may not provide much benefit and might even slow down the query due to the overhead of maintaining the index.

The Rating(rating) index did not affect the cost, which is likely because the query is performing an aggregation, and indexing individual ratings may not help with this kind of aggregate operation.

Indexing on Location(has_police_station) increased the cost significantly (more than 2x), which is unexpected because adding an index on has_police_station should ideally help with filtering based on whether a location has a police station. However, it's possible that indexing on a boolean column, which has low cardinality, resulted in poorer performance. The database likely ended up scanning a lot of data anyway, which worsened performance.

Given these results, I would likely not add any indexing for this query, since we tried indexing various attributes that are not primary keys but used the query, and this resulted in no improvement or degradation in performance. I think that since primary keys are indexed by default we chose to make the primary key for the Rating table (username, address) rather than assigning a rating_id, this already contributed to decent performance.

Query 3

```
mysql> explain analyze SELECT address, AVG(crash_severity_number) as average_crash_severity_rating
    -> FROM Pedestrian_Crash JOIN Apartment USING(Block) JOIN Location USING(Block)
    -> WHERE has_police_station = 'FALSE'
    -> GROUP BY address;
+--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------+
| EXPLAIN



                                             |
+--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------+
| -> Table scan on <temporary>  (actual time=219.869..220.102 rows=1411 loops=1)
    -> Aggregate using temporary table  (actual time=219.865..219.865 rows=1411 loops=1)
        -> Nested loop inner join  (cost=1831.12 rows=7254) (actual time=78.823..141.871 rows=104302 loops=1)
            -> Nested loop inner join  (cost=63.93 rows=154) (actual time=64.928..67.068 rows=660 loops=1)
                -> Filter: (Location.has_police_station = 0)  (cost=11.00 rows=10) (actual time=52.970..53.060 rows=52 loops=1)
                    -> Table scan on Location  (cost=11.00 rows=100) (actual time=52.960..53.000 rows=100 loops=1)
                -> Index lookup on Pedestrian_Crash using block (block=Location.`block`)  (cost=3.90 rows=15) (actual time=0.245..0.268 rows=13 loops=52)
            -> Covering index lookup on Apartment using block (block=Location.`block`)  (cost=6.78 rows=47) (actual time=0.033..0.102 rows=158 loops=660)
 |
+--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------+
```

```
mysql> create index location_station on Location(has_police_station);
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze SELECT address, AVG(crash_severity_number) as average_crash_severity_rating FROM Pedestrian_Crash JOIN Apartment USING(Block) JOIN Location USING(Block) W
RE has_police_station = 'FALSE' GROUP BY address;
+--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-----------------------+
| EXPLAIN



                                                                                                                  |
+--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-----------------------+
| -> Table scan on <temporary>  (actual time=115.680..115.859 rows=1411 loops=1)
    -> Aggregate using temporary table  (actual time=115.677..115.677 rows=1411 loops=1)
        -> Nested loop inner join  (cost=3393.96 rows=19992) (actual time=0.076..39.978 rows=104302 loops=1)
            -> Nested loop inner join  (cost=369.35 rows=425) (actual time=0.062..2.154 rows=660 loops=1)
                -> Filter: (Pedestrian_Crash.`block` is not null)  (cost=83.05 rows=818) (actual time=0.051..0.671 rows=818 loops=1)
                    -> Table scan on Pedestrian_Crash  (cost=83.05 rows=818) (actual time=0.049..0.576 rows=818 loops=1)
                -> Filter: (Location.has_police_station = 0)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=818)
                    -> Single-row index lookup on Location using PRIMARY (block=Pedestrian_Crash.`block`)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=818)
            -> Covering index lookup on Apartment using block (block=Pedestrian_Crash.`block`)  (cost=2.42 rows=47) (actual time=0.009..0.047 rows=158 loops=660)
 |
+--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
```

```
mysql> create index apartment_block on Apartment(block);
Query OK, 0 rows affected (0.19 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze SELECT address, AVG(crash_severity_number) as average_crash_severity_rating FROM Pedestrian_Crash JOIN Apartment USING(Block) JOIN Location USING(Block
RE has_police_station = 'FALSE' GROUP BY address;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------+
| EXPLAIN                                                                                                                                                                   |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------+
| -> Table scan on <temporary>  (actual time=219.462..219.777 rows=1411 loops=1)
    -> Aggregate using temporary table  (actual time=219.460..219.460 rows=1411 loops=1)
        -> Nested loop inner join  (cost=1223.14 rows=3845) (actual time=0.856..72.523 rows=104302 loops=1)
            -> Nested loop inner join  (cost=369.35 rows=82) (actual time=0.122..4.148 rows=660 loops=1)
                -> Filter: (Pedestrian_Crash.`block` is not null)  (cost=83.05 rows=818) (actual time=0.098..1.309 rows=818 loops=1)
                    -> Table scan on Pedestrian_Crash  (cost=83.05 rows=818) (actual time=0.095..1.155 rows=818 loops=1)
                -> Filter: (Location.has_police_station = 0)  (cost=0.25 rows=0.1) (actual time=0.003..0.003 rows=1 loops=818)
                    -> Single-row index lookup on Location using PRIMARY (block=Pedestrian_Crash.`block`)  (cost=0.25 rows=1) (actual time=0.002..0.003 rows=1 loops=818)
            -> Covering index lookup on Apartment using apartment_block (block=Pedestrian_Crash.`block`)  (cost=5.79 rows=47) (actual time=0.017..0.088 rows=158 loops=660)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
mysql> create index ped_block on Pedestrian_Crash(block);
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze SELECT address, AVG(crash_severity_number) as average_crash_severity_rating FROM Pedestrian_Crash JOIN Apartment USING(Block) JOIN Location USING(Block) WHE
RE has_police_station = 'FALSE' GROUP BY address;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------+
| EXPLAIN                                                                                                                                                                   |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------+
| -> Table scan on <temporary>  (actual time=118.136..118.327 rows=1411 loops=1)
    -> Aggregate using temporary table  (actual time=118.133..118.133 rows=1411 loops=1)
        -> Nested loop inner join  (cost=1223.14 rows=3845) (actual time=0.120..41.188 rows=104302 loops=1)
            -> Nested loop inner join  (cost=369.35 rows=82) (actual time=0.100..2.167 rows=660 loops=1)
                -> Filter: (Pedestrian_Crash.`block` is not null)  (cost=83.05 rows=818) (actual time=0.085..0.711 rows=818 loops=1)
                    -> Table scan on Pedestrian_Crash  (cost=83.05 rows=818) (actual time=0.084..0.616 rows=818 loops=1)
                -> Filter: (Location.has_police_station = 0)  (cost=0.25 rows=0.1) (actual time=0.002..0.002 rows=1 loops=818)
                    -> Single-row index lookup on Location using PRIMARY (block=Pedestrian_Crash.`block`)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=818)
            -> Covering index lookup on Apartment using apartment_block (block=Pedestrian_Crash.`block`)  (cost=5.79 rows=47) (actual time=0.009..0.047 rows=158 loops=660)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

In this query, we try indexing through three attributes: Location's has_police_station, Apartment's block, and Pedestrian_Crash's block.

1. For the first index design, we tried indexing through the location's has_police_station, which increased the cost by 1562.84, almost double the amount of the initial query. This might be because the cardinality of the column is pretty low: we only have two values and so indexing might not be the most efficient and a full table scan may be better.
2. For the second one, we tried indexing through Apartment's block, which improved the cost by decreasing it to 1223.14 from 1831.12. This might be because the block has a pretty high cardinality(we have lots of values it could be in this) which can help narrow the number of rows that need to be considered during the join, so less comparisons, and a bit faster.
3. For the third one, we tried indexing through Pedestrian_Crash's block, resulting in the same cost values as Apartment's block. This is because we don't have to look/scan through the entire table, we can just directly access the rows with the block values we want which increases efficiency and makes it faster. Also similarly, the cardinality is high so we can narrow the rows that we need during the join.

For the final index design, we can index on either the Pedestrian Crash Block attribute or Apartment Block attribute as it decreased the time and the cost.

Query 4

```
mysql> explain analyze WITH AvgWattagePerBlock AS ( SELECT block, AVG(wattage) AS avg_wattage FROM Streetlight GROUP BY block ), CrashesPerBlock AS ( SELECT block, COUNT(*) AS num
_crashes FROM Pedestrian_Crash GROUP BY block ), AverageValues AS ( SELECT (SELECT AVG(wattage) FROM Streetlight) AS overall_avg_wattage, (SELECT AVG(num_crashes) FROM CrashesPerB
lock) AS overall_avg_crashes ) SELECT l.block FROM Location l JOIN AvgWattagePerBlock w USING(block) JOIN CrashesPerBlock c USING(block) CROSS JOIN AverageValues a WHERE w.avg_wat
tage < a.overall_avg_wattage AND c.num_crashes > a.overall_avg_crashes;
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                                                                       |
|                                                                                                                                                                               |
|                                                                                                                                                                               |
|                                                                                                                                                                               |
|                                                                                                                                                                               |
|                                                                                                                                                                               |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=4524.91 rows=4180) (actual time=56.757..56.793 rows=5 loops=1)
    -> Nested loop inner join  (cost=135.41 rows=273) (actual time=0.045..0.102 rows=12 loops=1)
        -> Filter: ((c.num_crashes > '15.4340') and (c.`block` is not null))  (cost=245.89..39.99 rows=273) (actual time=0.021..0.043 rows=12 loops=1)
            -> Table scan on c  (cost=246.67..259.38 rows=818) (actual time=0.008..0.018 rows=53 loops=1)
                -> Materialize CTE CrashesPerBlock if needed (query plan printed elsewhere)  (cost=246.65..246.65 rows=818) (never executed)
        -> Single-row covering index lookup on l using PRIMARY (block=c.`block`)  (cost=0.25 rows=1) (actual time=0.004..0.005 rows=1 loops=12)
    -> Filter: (w.avg_wattage < '113.9480')  (cost=1283.12..11.51 rows=15) (actual time=4.724..4.724 rows=0 loops=12)
        -> Covering index lookup on w using <auto_key0> (block=c.`block`)  (actual time=4.723..4.723 rows=1 loops=12)
            -> Materialize CTE AvgWattagePerBlock  (cost=1371.85..1371.85 rows=4557) (actual time=55.751..55.751 rows=57 loops=1)
                -> Group aggregate: avg(Streetlight.wattage)  (cost=916.15 rows=4557) (actual time=30.460..55.595 rows=57 loops=1)
                    -> Index scan on Streetlight using block  (cost=460.45 rows=4557) (actual time=30.444..54.994 rows=4557 loops=1)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

CREATE INDEX idx_streetlight_wattage ON Streetlight(wattage);

```
-------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=4524.91 rows=4180) (actual time=5.876..5.908 rows=5 loops=1)
    -> Nested loop inner join  (cost=135.41 rows=273) (actual time=0.031..0.072 rows=12 loops=1)
        -> Filter: ((c.num_crashes > '15.4340') and (c.`block` is not null))  (cost=245.89..39.99 rows=273) (actual time=0.009..0.026 rows=12 loops=1)
            -> Table scan on c  (cost=246.67..259.38 rows=818) (actual time=0.002..0.010 rows=53 loops=1)
                -> Materialize CTE CrashesPerBlock if needed (query plan printed elsewhere)  (cost=246.65..246.65 rows=818) (never executed)
        -> Single-row covering index lookup on l using PRIMARY (block=c.`block`)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=12)
    -> Filter: (w.avg_wattage < '113.9480')  (cost=1283.12..11.51 rows=15) (actual time=0.486..0.486 rows=0 loops=12)
        -> Covering index lookup on w using <auto_key0> (block=c.`block`)  (actual time=0.485..0.486 rows=1 loops=12)
            -> Materialize CTE AvgWattagePerBlock  (cost=1371.85..1371.85 rows=4557) (actual time=5.809..5.809 rows=57 loops=1)
                -> Group aggregate: avg(Streetlight.wattage)  (cost=916.15 rows=4557) (actual time=0.123..5.740 rows=57 loops=1)
                    -> Index scan on Streetlight using block  (cost=460.45 rows=4557) (actual time=0.120..5.236 rows=4557 loops=1)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

CREATE INDEX idx_streetlight_block ON Streetlight(block);

```
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=4524.91 rows=4180) (actual time=5.507..5.540 rows=5 loops=1)
    -> Nested loop inner join  (cost=135.41 rows=273) (actual time=0.042..0.082 rows=12 loops=1)
        -> Filter: ((c.num_crashes > '15.4340') and (c.`block` is not null))  (cost=245.89..39.99 rows=273) (actual time=0.020..0.036 rows=12 loops=1)
            -> Table scan on c  (cost=246.67..259.38 rows=818) (actual time=0.007..0.015 rows=53 loops=1)
                -> Materialize CTE CrashesPerBlock if needed (query plan printed elsewhere)  (cost=246.65..246.65 rows=818) (never executed)
        -> Single-row covering index lookup on l using PRIMARY (block=c.`block`)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=12)
    -> Filter: (w.avg_wattage < '113.9480')  (cost=1283.12..11.51 rows=15) (actual time=0.454..0.454 rows=0 loops=12)
        -> Covering index lookup on w using <auto_key0> (block=c.`block`)  (actual time=0.454..0.454 rows=1 loops=12)
            -> Materialize CTE AvgWattagePerBlock  (cost=1371.85..1371.85 rows=4557) (actual time=5.431..5.431 rows=57 loops=1)
                -> Group aggregate: avg(Streetlight.wattage)  (cost=916.15 rows=4557) (actual time=0.145..5.371 rows=57 loops=1)
                    -> Index scan on Streetlight using idx_streetlight_block  (cost=460.45 rows=4557) (actual time=0.140..4.913 rows=4557 loops=1)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

CREATE INDEX idx_pedestrian_crash_block ON Pedestrian_Crash(block);

```
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=4524.91 rows=4180) (actual time=5.516..5.550 rows=5 loops=1)
    -> Nested loop inner join  (cost=135.41 rows=273) (actual time=0.026..0.069 rows=12 loops=1)
        -> Filter: ((c.num_crashes > '15.4340') and (c.`block` is not null))  (cost=245.89..39.99 rows=273) (actual time=0.011..0.029 rows=12 loops=1)
            -> Table scan on c  (cost=246.67..259.38 rows=818) (actual time=0.004..0.013 rows=53 loops=1)
                -> Materialize CTE CrashesPerBlock if needed (query plan printed elsewhere)  (cost=246.65..246.65 rows=818) (never executed)
        -> Single-row covering index lookup on l using PRIMARY (block=c.`block`)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=12)
    -> Filter: (w.avg_wattage < '113.9480')  (cost=1283.12..11.51 rows=15) (actual time=0.456..0.456 rows=0 loops=12)
        -> Covering index lookup on w using <auto_key0> (block=c.`block`)  (actual time=0.456..0.456 rows=1 loops=12)
            -> Materialize CTE AvgWattagePerBlock  (cost=1371.85..1371.85 rows=4557) (actual time=5.442..5.442 rows=57 loops=1)
                -> Group aggregate: avg(Streetlight.wattage)  (cost=916.15 rows=4557) (actual time=0.132..5.350 rows=57 loops=1)
                    -> Index scan on Streetlight using idx_streetlight_block  (cost=460.45 rows=4557) (actual time=0.128..4.869 rows=4557 loops=1)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

For the first index design, we tried indexing the wattage attribute for the streetlights table. We thought this would be a good idea because we calculate the average wattage for each block. We surprisingly did not see any improvement in performance, and the reason is probably

because the database decided that scanning the entire table was more efficient than using the index due to the nature of the aggregation.

For the second index design, we tried indexing the streetlight block attribute. We thought this was a good index to try because it is involved in a JOIN operation. The reason why we didn't find any difference in the results is probably because the block is a foreign key and foreign keys are automatically indexed by the database system.

For the third index design, we tried indexing the block attribute in pedestrian crash because it is involved in a JOIN operation. Similar to the previous reasons, the reason why we didn't find any difference in the results is probably because the block is a foreign key and foreign keys are oftentimes automatically indexed by the database system.

Overall, we are choosing the original database design with no manual indexing additions. If we were to improve the cost of our query performance, we would probably focus on making the query itself faster. For example, JOINS are an expensive operation so instead of using a CROSS JOIN, we can probably just have 2 additional subqueries that would go inside the where clause.