

Here is a picture showing the terminal of the GCP with the databases with the tables we implemented. Our database is located on the Google GCP.

```
mysql> show tables;
+-----+
| Tables_in_project |
+-----+
| Apartments        |
| CrimeEvents        |
| CrimeTypes         |
| Neighborhoods      |
| Prefers            |
| Users              |
+-----+
6 rows in set (0.01 sec)
```

DDL Commands For Creating the Table

Users:

```
CREATE TABLE Users (
  UserId VARCHAR(255) PRIMARY KEY,
  Email VARCHAR(255) UNIQUE,
  Password VARCHAR(255),
  Gender VARCHAR(255),
  Age INT DEFAULT -1
);
```

Prefers:

```
CREATE TABLE Prefers(
  ID VARCHAR(255),
  Interest VARCHAR(255),
  FOREIGN KEY (ID) REFERENCES Users(UserId),
  FOREIGN KEY (Interest) REFERENCES Neighborhoods(NeighborhoodName),
  PRIMARY KEY (ID, Interest)
);
```

CRIME EVENTS:

```
CREATE TABLE CrimeEvents (
  CrimeId INTEGER PRIMARY KEY,
```

```

CrimeNeighborhood VARCHAR(255),
FOREIGN KEY (CrimeNeighborhood) REFERENCES Neighborhoods(NeighborhoodName),
Date DATETIME,
LocationDescription VARCHAR(255),
Block VARCHAR(255),
PrimaryType VARCHAR(255),
Arrested BOOLEAN
);

```

Neighborhoods:

```

CREATE TABLE Neighborhoods (
    NeighborhoodName VARCHAR(255) PRIMARY KEY,
    AverageRent INTEGER ,
    AverageAge INTEGER ,
    Demographic VARCHAR(255)
);

```

Apartments:

```

CREATE TABLE Apartments (
    ApartmentId INTEGER PRIMARY KEY,
    Neighborhood VARCHAR(255),
    FOREIGN KEY (Neighborhood) REFERENCES Neighborhoods(NeighborhoodName),
    RenterCompanyName VARCHAR(255),
    Rating INTEGER,
    Cost INTEGER
);

```

```

mysql> show tables;
+-----+
| Tables_in_project |
+-----+
| Apartments        |
| CrimeEvents       |
| CrimeTypes        |
| Neighborhoods     |
| Prefers           |
| Users             |
+-----+
6 rows in set (0.01 sec)

```

Entry into the Table Implementation

We decided to implement implement entries for Users, Crime Type and Crime Event

For Table Users, some users are inserted into the table.

```
INSERT INTO Users (UserId, Email, Password, Gender, Age)
VALUES ('user01', 'paul@google.com', 'password1', 'male', 32);
VALUES ('user02', 'ethan@google.com', 'password2', 'male', 23);
VALUES ('user03', 'alex@google.com', 'password3', 'male', 45);
VALUES ('user04', 'linda@google.com', 'password4', 'female', 37);
VALUES ('user05', 'mike@google.com', 'password5', 'male', 40);
```

For Table CrimeType, some CrimeTypes are inserted into the table.

```
INSERT INTO CrimeTypes (Crimeld, PrimaryType , Arrested)
VALUES (12589893, 'SEX OFFENSE', FALSE);
VALUES (12592454, 'OTHER OFFENSE', FALSE);
VALUES (12601676, 'OFFENSE INVOLVING CHILDREN', TRUE);
```

For Table CrimeEvent, some CrimeEvent are inserted into the table.

```
INSERT INTO CrimeEvent(Crimeld, Date, LocationDescription, Block)
VALUES (12589893, 1/11/2022 3:00, RESIDENCE, 087XX S KINGSTON AVE);
VALUES (12592454, 1/14/2022 15:55, RESIDENCE, 067XX S MORGAN ST);
VALUES (12601676, 1/13/2022 16:00, STREET, 031XX W AUGUSTA BLVD);
```

SELECT * FROM CrimeEvents;

```
10-00 00:00:00 | 026XX N Burling St
100 00:00:00 | RESIDENCE | 070XX W IMLAY ST
100 00:00:00 | RESIDENCE | 066XX S LOWE AVE
10-00 00:00:00 | 006XX W MADISON ST
1-00 00:00:00 | RESIDENCE | 101XX S YATES AVE
10-00-00 00:00:00 | APARTMENT | 038XX S LAKE PARK AVE
100-00 00:00:00 | RESIDENCE | 065XX S SANGAMON ST
10 00:00:00 | APARTMENT | 011XX E 82ND ST
10-00-00 00:00:00 | OTHER (SPECIFY) | 035XX N GREENVIEW AVE
10-00 00:00:00 | RESIDENCE | 060XX S AUSTIN AVE
10-00 00:00:00 | RESIDENCE | 117XX S JUSTINE ST
10-00 00:00:00 | 109XX S VERNON AVE
100-00 00:00:00 | AUTO / BOAT / RV DEALERSHIP | 019XX W PERSHING RD
1-00-00 00:00:00 | RESIDENCE | 080XX S EXCHANGE AVE
10-00-00 00:00:00 | RESIDENCE - PORCH / HALLWAY | 037XX W FULLERTON AVE
1-00-00 00:00:00 | HOSPITAL BUILDING / GROUNDS | 057XX W ROOSEVELT RD
1-00-00 00:00:00 | RESIDENCE | 054XX W ROSEDALE AVE
1-00-00 00:00:00 | COMMERCIAL / BUSINESS OFFICE | 042XX S KILDARE BLVD
100-00 00:00:00 | APARTMENT | 021XX N KILDARE AVE
100-00 00:00:00 | RESTAURANT | 057XX N CENTRAL AVE
10-00 00:00:00 | STREET | 031XX N HALSTED ST
10-00-00 00:00:00 | APARTMENT | 021XX S PRINCETON AVE
100 00:00:00 | SIDEWALK | 031XX N BROADWAY
1-00 00:00:00 | ABANDONED BUILDING | 021XX W GRAND AVE
100-00 00:00:00 | STREET | 009XX W RANDOLPH ST
100-00 00:00:00 | OTHER (SPECIFY) | 077XX S EMERALD AVE
10-00 00:00:00 | APARTMENT | 016XX W PIERCE AVE
1-00 00:00:00 | APARTMENT | 056XX W NORTH AVE
1-00-00 00:00:00 | COMMERCIAL / BUSINESS OFFICE | 032XX W LAWRENCE AVE
10-00-00 00:00:00 | 035XX N CLAREMONT AVE
+-----+-----+
1001 rows in set (0.00 sec)
```

```
mysql> show tables;
+-----+
| Tables_in_project |
+-----+
| Apartments        |
| CrimeEvents        |
| CrimeTypes         |
| Neighborhoods      |
| Prefers            |
| Users              |
+-----+
6 rows in set (0.01 sec)

mysql> SELECT COUNT(*) FROM USERS;
ERROR 1146 (42S02): Table 'project.USERS' doesn't exist
mysql> SELECT COUNT(*) FROM Users;
+-----+
| COUNT(*) |
+-----+
|      644 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM CrimeTypes;
+-----+
| COUNT(*) |
+-----+
|     1001 |
+-----+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM CrimeEvents;
+-----+
| COUNT(*) |
+-----+
|     1001 |
+-----+
1 row in set (0.00 sec)
```

Users is a CSV with 1001 entries in it, but it is currently an error with showing 644.
 Crime types and Crime events are working as intended.

Advanced Queries

The Following queries have empty sets as the data was automatically generated and thus not fitting into the criteria of the Advanced Queries.

1. Get the neighborhood name and average age of neighborhoods with the most crimes

```
SELECT n.NeighborhoodName, n.AverageAge
FROM Neighborhoods n
WHERE n.NeighborhoodName in
      (SELECT ce.CrimeNeighborhood
       FROM CrimeEvents ce
       WHERE ce.Arrested = 1
       GROUP BY ce.CrimeNeighborhood
       ORDER BY COUNT(*) DESC)
LIMIT 15;
```

New:

```
mysql> SELECT n.NeighborhoodName, n.AverageAge
-> FROM Neighborhoods n
-> WHERE n.NeighborhoodName in
-> (SELECT ce.CrimeNeighborhood
-> FROM CrimeEvents ce
-> WHERE ce.Arrested = 1
-> GROUP BY ce.CrimeNeighborhood
-> ORDER BY COUNT(*) DESC)
-> LIMIT 15;
```

NeighborhoodName	AverageAge
Albany Park	36
Andersonville	68
Archer Heights	53
Armour Square	30
Ashburn	65
Auburn Gresham	57
Austin	26
Avalon Park	61
Avondale	55
Belmont Cragin	45
Beverly	64
Bridgeport	23
Brighton Park	62
Bronzeville	28
Bucktown	29

```
15 rows in set (0.01 sec)
```

Old:

1. Find what crime type was most committed between 10 and 11 AM.

```
SELECT CrimeTypes.PrimaryType, COUNT(CrimeEvents.CrimeId) AS CrimeNum
FROM CrimeTypes
JOIN CrimeEvents ON CrimeTypes.CrimeId = CrimeEvents.CrimeId
WHERE TIME(CrimeEvents.Date) BETWEEN '00:10:00' AND '00:11:00'
GROUP BY CrimeTypes.PrimaryType
ORDER BY CrimeNum DESC
LIMIT 15;
```

```
mysql> SELECT CrimeTypes.PrimaryType, COUNT(CrimeEvents.CrimeId) AS CrimeNum
-> FROM CrimeTypes
-> JOIN CrimeEvents ON CrimeTypes.CrimeId = CrimeEvents.CrimeId
-> WHERE TIME(CrimeEvents.Date) BETWEEN '00:10:00' AND '00:11:00'
-> GROUP BY CrimeTypes.PrimaryType
-> ORDER BY CrimeNum DESC
-> LIMIT 15;
Empty set (0.00 sec)
```

This Query counts the number of crimes between the times of 10 and 11 AM and returns the most common crime type in that span of time.

2. Get 15 neighborhoods where there have been fewer than 10 arrests

```
SELECT n.NeighborhoodName
FROM Neighborhoods n JOIN CrimeEvents ce ON n.NeighborhoodName =
ce.CrimeNeighborhood
WHERE ce.Arrested = 1
GROUP BY n.NeighborhoodName
HAVING COUNT(*) < 10
LIMIT 15;
```

NEW:

```
mysql> SELECT n.NeighborhoodName
-> FROM Neighborhoods n JOIN CrimeEvents ce ON n.NeighborhoodName = ce.CrimeNeighborhood
-> WHERE ce.Arrested = 1
-> GROUP BY n.NeighborhoodName
-> HAVING COUNT(*) < 10
-> LIMIT 15;

+-----+
| NeighborhoodName |
+-----+
| South Lawndale (Little Village) |
| Calumet Heights |
| Printer's Row |
| The Loop |
| Chatham |
| Wicker Park |
| McKinley Park |
| Archer Heights |
| Lawndale |
| Lower West Side |
| Oakland |
| Gold Coast |
| Margate Park |
| Auburn Gresham |
| Washington Heights |
+-----+
15 rows in set (0.00 sec)
```

OLD:

```
mysql> SELECT n.NeighborhoodName
-> FROM Neighborhoods n NATURAL JOIN CrimeEvents ce NATURAL JOIN CrimeTypes ct
-> WHERE ct.Arrested = 0
-> GROUP BY n.NeighborhoodName
-> HAVING COUNT(*) < 20
-> LIMIT 15;
Empty set (0.08 sec)
```

3. Get 15 neighborhoods with an average rent less than 2000 dollars and an average rating of 5 across all renter companies

```
(SELECT n.NeighborhoodName
FROM Neighborhoods n
WHERE n.AverageRent < 2000
```

INTERSECT

```
SELECT a.Neighborhood
FROM Apartments a
GROUP BY a.Neighborhood
HAVING AVG(Rating) > 5)
```


LIMIT 15;

NEW:

```
mysql> (SELECT n.NeighborhoodName
-> FROM Neighborhoods n
-> WHERE n.AverageRent < 2000
->
-> INTERSECT
->
-> SELECT a.Neighborhood
-> FROM Apartments a
-> GROUP BY a.Neighborhood
-> HAVING AVG(Rating) > 5)
-> LIMIT 15;
```

NeighborhoodName
Ashburn
Avalon Park
Bridgeport
Bronzeville
Cabrini-Green
Chatham
Clearing
East Garfield Park
East Side
Edgewater
Englewood
Gage Park
Gold Coast
Hermosa
Humboldt Park

15 rows in set (0.01 sec)

OLD:

```
mysql> (SELECT n.NeighborhoodName
-> FROM Neighborhoods n
-> WHERE n.AverageRent < 1000
->
-> INTERSECT
->
-> SELECT a.Neighborhood
-> FROM Apartments a
-> GROUP BY a.Neighborhood
-> HAVING AVG(Rating) > 7.5)
-> LIMIT 15;
Empty set (0.01 sec)
```

4. **Get neighborhoods with an average age under 50 and where there are more males than females interested in it.**

```
SELECT DISTINCT n.NeighborhoodName
FROM Prefers p
JOIN Users u ON p.ID = u.UserId
JOIN Neighborhoods n ON p.Interest = n.NeighborhoodName
WHERE n.AverageAge < 50
AND n.NeighborhoodName IN
    (SELECT pref.Interest
     FROM Prefers pref JOIN Users us ON pref.ID = us.UserId
     WHERE us.Gender = 'Male'
     GROUP BY pref.Interest
     HAVING COUNT(*) >
        (SELECT COUNT(*)
         FROM Prefers pref2 JOIN Users us2 ON pref2.ID = us2.UserId
         WHERE pref2.Interest = pref.Interst AND us2.Gender = 'Female'))
LIMIT 15;
```

New:

```
mysql> SELECT DISTINCT n.NeighborhoodName
-> FROM Prefers p
-> JOIN Users u ON p.ID = u.UserId
-> JOIN Neighborhoods n ON p.Interest = n.NeighborhoodName
-> WHERE n.AverageAge < 50
-> AND n.NeighborhoodName IN
->     (SELECT pref.Interest
->      FROM Prefers pref
->      JOIN Users us ON pref.ID = us.UserId
->      WHERE us.Gender = 'Male'
->      GROUP BY pref.Interest
->      HAVING COUNT(*) >
->           (SELECT COUNT(*)
->            FROM Prefers pref2
->            JOIN Users us2 ON pref2.ID = us2.UserId
->            WHERE pref2.Interest = pref.Interest AND us2.Gender = 'Female'))
-> LIMIT 15;
+-----+
| NeighborhoodName |
+-----+
| Bridgeport      |
| Bronzeville     |
| Bucktown        |
| Clearing         |
| Forest Glen     |
| Garfield Ridge  |
| Greektown       |
| Hegewisch       |
| McKinley Park   |
| Morgan Park     |
| Pullman         |
| The Loop        |
+-----+
12 rows in set (0.00 sec)
```

Old:

```
mysql> SELECT n.NeighborhoodName
-> FROM Prefers p NATURAL JOIN Users u NATURAL JOIN Neighborhoods n
-> WHERE n.AverageAge < 25 AND n.NeighborhoodName IN
->      (SELECT pref.NeighborhoodName
->      FROM Prefers pref
->      NATURAL JOIN Users us
->      WHERE us.Gender = 'Male'
->      GROUP BY pref.NeighborhoodName
->      HAVING COUNT(*) > 20)
->
-> LIMIT 15;
Empty set (0.00 sec)
```

Indexing Analysis

Advanced Query 1

Using this query, we will try to improve the costs with indexing.

Explain Analyze

```
SELECT n.NeighborhoodName, n.AverageAge
FROM Neighborhoods n
WHERE n.NeighborhoodName in
      (SELECT ce.CrimeNeighborhood
      FROM CrimeEvents ce
      WHERE ce.Arrested = 1
      GROUP BY ce.CrimeNeighborhood
      ORDER BY COUNT(*) DESC)
LIMIT 15;
```

We first use Explain Analyze to find the performance of the original query.

```

-----+
| -> Limit: 15 row(s) (cost=10.55 rows=15) (actual time=2.024..2.055 rows=15 loops=1)
   -> Filter: <in_optimizer>(n.NeighborhoodName,n.NeighborhoodName in (select #2)) (cost=10.55 rows=103) (actual time=2.023..2.052 row
s=15 loops=1)
       -> Table scan on n (cost=10.55 rows=103) (actual time=0.068..0.074 rows=15 loops=1)
       -> Select #2 (subquery in condition; run only once)
           -> Filter: ((n.NeighborhoodName = '<materialized_subquery>'.CrimeNeighborhood)) (cost=123.85..123.85 rows=1) (actual time=0
.123..0.123 rows=1 loops=16)
               -> Limit: 1 row(s) (cost=123.75..123.75 rows=1) (actual time=0.122..0.122 rows=1 loops=16)
                   -> Index lookup on <materialized_subquery> using <auto_distinct_key> (CrimeNeighborhood=n.NeighborhoodName) (actual
time=0.122..0.122 rows=1 loops=16)
                       -> Materialize with deduplication (cost=123.75..123.75 rows=100) (actual time=1.935..1.935 rows=100 loops=1)
                           -> Group (no aggregates) (cost=113.75 rows=100) (actual time=0.125..1.867 rows=100 loops=1)
                               -> Filter: (ce.Arrested = 1) (cost=103.75 rows=100) (actual time=0.113..1.684 rows=678 loops=1)
                                   -> Index scan on ce using CrimeNeighborhood (cost=103.75 rows=1000) (actual time=0.111..1.518 rows=
1000 loops=1)
|

```

The Cost is 10.55, Time is 2.024-2.055 seconds

We then apply 3 indexing methods to try and attempt to improve our runtime:

1. Index on CrimeEvent table for the Crimeld

This index will speed up the JOIN operation between CrimeType and CrimeEvent tables, as it will quickly find the corresponding crimes in the CrimeType Table based on Crimeld.

mysql> CREATE INDEX idxCrimeld ON CrimeEvents (Crimeld)

After implementing these changes, we noticed an improvement

```

-----+
| -> Limit: 15 row(s) (cost=7.42 rows=15) (actual time=1.795..1.832 rows=15 loops=1)
   -> Filter: <in_optimizer>(n.NeighborhoodName,n.NeighborhoodName in (select #2)) (cost=10.55 rows=103) (actual time=1.794..1.830 row
s=15 loops=1)
       -> Table scan on n (cost=7.42 rows=103) (actual time=0.055..0.061 rows=15 loops=1)
       -> Select #2 (subquery in condition; run only once)
           -> Filter: ((n.NeighborhoodName = '<materialized_subquery>'.CrimeNeighborhood)) (cost=225.72..225.72 rows=1) (actual time=0
.109..0.109 rows=1 loops=16)
               -> Limit: 1 row(s) (cost=225.62..225.62 rows=1) (actual time=0.108..0.108 rows=1 loops=16)
                   -> Index lookup on <materialized_subquery> using <auto_distinct_key> (CrimeNeighborhood=n.NeighborhoodName) (actual
time=0.108..0.108 rows=1 loops=16)
                       -> Materialize with deduplication (cost=225.62..225.62 rows=678) (actual time=1.715..1.715 rows=100 loops=1)
                           -> Table scan on <temporary> (cost=146.87..157.82 rows=678) (actual time=1.661..1.675 rows=100 loops=1)
                               -> Temporary table with deduplication (cost=146.85..146.85 rows=678) (actual time=1.658..1.658 rows=100
loops=1)
                                   -> Index lookup on ce using idxArrested (Arrested=1) (cost=79.05 rows=678) (actual time=0.084..1.23
7 rows=678 loops=1)
|

```

The Cost is 7.42, Time is 1.795-1.832 seconds

Analysis: Since the CrimeEvents table is joined with the CrimeType table using Crimeld, having an index on Crimeld in CrimeEvents will help speed up the joining process of the two. The database would find the corresponding Crimeld in the CrimeTypes table making the join operation faster.

2. Index on Neighborhood table for the AverageAge

mysql> CREATE INDEX idxAverageAge ON Neighborhoods (AverageAge)

After implementing these changes, we did not notice much of a change in the overall cost, but the time took a large decrease.

```

-----+-----
| -> Limit: 15 row(s) (cost=10.55 rows=15) (actual time=1.392..1.427 rows=15 loops=1)
| -> Filter: <in_optimizer>(n.NeighborhoodName,n.NeighborhoodName in (select #2)) (cost=10.55 rows=103) (actual time=1.391..1.425 row
s=15 loops=1)
|   -> Table scan on n (cost=10.55 rows=103) (actual time=0.062..0.067 rows=15 loops=1)
|   -> Select #2 (subquery in condition; run only once)
|     -> Filter: ((n.NeighborhoodName = '<materialized_subquery>'.CrimeNeighborhood)) (cost=225.72..225.72 rows=1) (actual time=0
.084..0.084 rows=1 loops=16)
|       -> Limit: 1 row(s) (cost=225.62..225.62 rows=1) (actual time=0.083..0.083 rows=1 loops=16)
|         -> Index lookup on <materialized_subquery> using <auto_distinct_key> (CrimeNeighborhood=n.NeighborhoodName) (actual
time=0.083..0.083 rows=1 loops=16)
|           -> Materialize with deduplication (cost=225.62..225.62 rows=678) (actual time=1.314..1.314 rows=100 loops=1)
|             -> Table scan on <temporary> (cost=146.87..157.82 rows=678) (actual time=1.262..1.274 rows=100 loops=1)
|               -> Temporary table with deduplication (cost=146.85..146.85 rows=678) (actual time=1.259..1.259 rows=100
loops=1)
|                 -> Index lookup on ce using idxArrested (Arrested=1) (cost=79.05 rows=678) (actual time=0.089..1.02
0 rows=678 loops=1)
|
|

```

Analysis: The original cost sat at an 10.55 and remained at 10.55 and the time is 1.392-1.427.

3. Index on CrimeEvents table for the Arrested

mysql> CREATE INDEX idxArrested ON CrimeEvents (Arrested)

After implementing these changes, we noticed that the cost was the same and the time was slightly shorter.

```

-----+-----
| -> Limit: 15 row(s) (cost=10.55 rows=15) (actual time=1.418..1.448 rows=15 loops=1)
| -> Filter: <in_optimizer>(n.NeighborhoodName,n.NeighborhoodName in (select #2)) (cost=10.55 rows=103) (actual time=1.416..1.446 row
s=15 loops=1)
|   -> Table scan on n (cost=10.55 rows=103) (actual time=0.050..0.054 rows=15 loops=1)
|   -> Select #2 (subquery in condition; run only once)
|     -> Filter: ((n.NeighborhoodName = '<materialized_subquery>'.CrimeNeighborhood)) (cost=225.72..225.72 rows=1) (actual time=0
.086..0.086 rows=1 loops=16)
|       -> Limit: 1 row(s) (cost=225.62..225.62 rows=1) (actual time=0.085..0.085 rows=1 loops=16)
|         -> Index lookup on <materialized_subquery> using <auto_distinct_key> (CrimeNeighborhood=n.NeighborhoodName) (actual
time=0.085..0.085 rows=1 loops=16)
|           -> Materialize with deduplication (cost=225.62..225.62 rows=678) (actual time=1.350..1.350 rows=100 loops=1)
|             -> Table scan on <temporary> (cost=146.87..157.82 rows=678) (actual time=1.297..1.310 rows=100 loops=1)
|               -> Temporary table with deduplication (cost=146.85..146.85 rows=678) (actual time=1.295..1.295 rows=100
loops=1)
|                 -> Index lookup on ce using idxArrested (Arrested=1) (cost=79.05 rows=678) (actual time=0.086..1.01
2 rows=678 loops=1)
|
|

```

The Cost remains 10.55, Time is 1.418-1.448 seconds

Final Index Design would be selecting the first indexing of the CrimeEvent table for the CrimeId as it led to the greatest decrease in cost out of the three attempted indexing.

Advanced Query 2

Using this query, we will try to improve the costs with indexing.

Explain Analyze

SELECT n.NeighborhoodName

FROM Neighborhoods n JOIN CrimeEvents ce ON n.NeighborhoodName =

ce.CrimeNeighborhood

WHERE ce.Arrested = 1

GROUP BY n.NeighborhoodName

```
HAVING COUNT(*) < 10
LIMIT 15;
```

We first use Explain Analyze to find the performance of the original query.

```
| EXPLAIN
+-----+
|
+-----+
| -> Limit: 15 row(s) (cost=436.62 rows=15) (actual time=0.158..0.555 rows=15 loops=1)
|   -> Filter: (count(0) < 10) (cost=436.62 rows=691) (actual time=0.157..0.553 rows=15 loops=1)
|     -> Group aggregate: count(0) (cost=436.62 rows=691) (actual time=0.156..0.549 rows=17 loops=1)
|       -> Nested loop inner join (cost=367.48 rows=691) (actual time=0.116..0.514 rows=117 loops=1)
|         -> Covering index scan on n using idxNeighborhood (cost=10.55 rows=103) (actual time=0.051..0.054 rows=18 loops=1)
|         -> Filter: (ce.Arrested = 1) (cost=2.48 rows=7) (actual time=0.022..0.025 rows=6 loops=18)
|           -> Index lookup on ce using CrimeNeighborhood (CrimeNeighborhood=n.NeighborhoodName) (cost=2.48 rows=10) (actual time=0.021..0.024 rows=9 loops=18)
|
+-----+
```

This query has an overall cost of 436.62 and a time of 0.158-0.555 seconds

We then apply 3 indexing methods to try and attempt to improve our runtime:

1. Index on CrimeEvents table for the Arrested

```
mysql> CREATE INDEX idxArrested ON CrimeEvents (Arrested)
```

After implementing these changes, we noticed that the cost was the same and the time was slightly shorter.

```
+-----+
|
+-----+
| -> Limit: 15 row(s) (cost=436.62 rows=15) (actual time=0.202..0.728 rows=15 loops=1)
|   -> Filter: (count(0) < 10) (cost=436.62 rows=691) (actual time=0.200..0.726 rows=15 loops=1)
|     -> Group aggregate: count(0) (cost=436.62 rows=691) (actual time=0.199..0.722 rows=17 loops=1)
|       -> Nested loop inner join (cost=367.48 rows=691) (actual time=0.163..0.687 rows=117 loops=1)
|         -> Covering index scan on n using idxNeighborhood (cost=10.55 rows=103) (actual time=0.103..0.106 rows=18 loops=1)
|         -> Filter: (ce.Arrested = 1) (cost=2.48 rows=7) (actual time=0.028..0.032 rows=6 loops=18)
|           -> Index lookup on ce using CrimeNeighborhood (CrimeNeighborhood=n.NeighborhoodName) (cost=2.48 rows=10) (actual time=0.028..0.031 rows=9 loops=18)
|
+-----+
```

The Cost remains 436.62, Time is 0.202-0.728 seconds

2. Index on Neighborhood table for the NeighborhoodName

```
mysql> CREATE INDEX idxNeighborhoodName ON Neighborhoods
(NeighborhoodName)
```

After implementing these changes, we did not notice any change in the overall cost.

```

EXPLAIN
|
-----+
-> Limit: 15 row(s) (cost=436.62 rows=15) (actual time=0.210..0.868 rows=15 loops=1)
  -> Filter: (count(0) < 10) (cost=436.62 rows=691) (actual time=0.209..0.865 rows=15 loops=1)
    -> Group aggregate: count(0) (cost=436.62 rows=691) (actual time=0.207..0.860 rows=17 loops=1)
      -> Nested loop inner join (cost=367.48 rows=691) (actual time=0.141..0.798 rows=117 loops=1)
        -> Covering index scan on n using idxNeighborhood (cost=10.55 rows=103) (actual time=0.067..0.074 rows=18 loops=1)
        -> Filter: (ce.Arrested = 1) (cost=2.48 rows=7) (actual time=0.033..0.039 rows=6 loops=18)
          -> Index lookup on ce using CrimeNeighborhood (CrimeNeighborhood=n.NeighborhoodName) (cost=2.48 rows=10) (actual time=0.033..0.037 rows=9 loops=18)
|

```

The Cost remains 436.62, Time is 0.210-0.868 seconds

3. Index on CrimeEvent table for the ce.CrimeNeighborhood

This index will speed up the JOIN operation between Neighborhoods and CrimeEvent tables, as it will quickly find the ce.CrimeNeighborhood

```
mysql> CREATE INDEX idxCrimeNeighborhood ON CrimeEvents
(CrimeNeighborhood)
```

After implementing these changes, we noticed an improvement in the cost

```

-----+
| --> Limit: 15 row(s) (cost=354.87 rows=15) (actual time=0.156..0.669 rows=15 loops=1)
|   --> Filter: (count(0) < 10) (cost=354.87 rows=691) (actual time=0.154..0.666 rows=15 loops=1)
|     --> Group aggregate: count(0) (cost=354.87 rows=691) (actual time=0.153..0.662 rows=17 loops=1)
|       --> Nested loop inner join (cost=277.48 rows=691) (actual time=0.115..0.621 rows=117 loops=1)
|         --> Covering index scan on n using idxNeighborhood (cost=10.55 rows=103) (actual time=0.052..0.057 rows=18 loops=1)
|         --> Filter: (ce.Arrested = 1) (cost=2.48 rows=7) (actual time=0.024..0.031 rows=6 loops=18)
|           --> Index lookup on ce using CrimeNeighborhood (CrimeNeighborhood=n.NeighborhoodName) (cost=2.48 rows=10) (actual ti
me=0.024..0.029 rows=9 loops=18)
|
+-----+

```

4. The Cost is now 354.87, Time is 0.156-0.669 seconds

Final Index Design would be selecting the first indexing of the CrimeEvent table for the ce.CrimeNeighborhood as it led to the greatest decrease in cost out of the three attempted indexing.

Advanced Query 3

Using this query, we will try to improve the costs with indexing.

```
Explain analyze
(SELECT n.NeighborhoodName
FROM Neighborhoods n
WHERE n.AverageRent < 2000
```

INTERSECT

```
SELECT a.Neighborhood
FROM Apartments a
GROUP BY a.Neighborhood
HAVING AVG(Rating) > 5)
LIMIT 15;
```

We first use Explain Analyze to find the performance of the original query.

```
-----+
| -> Limit: 15 row(s) (cost=294.72..295.91 rows=15) (actual time=2.606..2.609 rows=15 loops=1)
|   -> Table scan on <intersect temporary> (cost=294.72..297.56 rows=34) (actual time=2.605..2.608 rows=15 loops=1)
|     -> Intersect materialize with deduplication (cost=294.63..294.63 rows=34) (actual time=2.603..2.603 rows=48 loops=1)
|       -> Filter: (n.AverageRent < 2000) (cost=10.55 rows=34) (actual time=0.063..0.087 rows=48 loops=1)
|         -> Table scan on n (cost=10.55 rows=103) (actual time=0.061..0.078 rows=103 loops=1)
|       -> Filter: (avg(a.Rating) > 5) (cost=280.65 rows=1392) (actual time=0.165..2.417 rows=74 loops=1)
|         -> Group aggregate: avg(a.Rating) (cost=280.65 rows=1392) (actual time=0.157..2.384 rows=96 loops=1)
|           -> Index scan on a using Neighborhood (cost=141.45 rows=1392) (actual time=0.143..1.956 rows=1391 loops=1)
|
|-----+
|
```

This query has an overall cost of 294.72-295.91 and a time of 2.606-2.609 seconds

We then apply 3 indexing methods to try and attempt to improve our runtime:

1. Index on Neighborhoods table for the Neighborhood

```
mysql> CREATE INDEX idxNeighborhood ON Neighborhoods (Neighborhood)
```

After implementing these changes, we noticed that the cost was the same.

```
+-----+
| -> Limit: 15 row(s) (cost=294.72..295.91 rows=15) (actual time=2.353..2.356 rows=15 loops=1)
|   -> Table scan on <intersect temporary> (cost=294.72..297.56 rows=34) (actual time=2.352..2.354 rows=15 loops=1)
|     -> Intersect materialize with deduplication (cost=294.63..294.63 rows=34) (actual time=2.350..2.350 rows=48 loops=1)
|       -> Filter: (n.AverageRent < 2000) (cost=10.55 rows=34) (actual time=0.052..0.076 rows=48 loops=1)
|         -> Table scan on n (cost=10.55 rows=103) (actual time=0.050..0.067 rows=103 loops=1)
|       -> Filter: (avg(a.Rating) > 5) (cost=280.65 rows=1392) (actual time=0.140..0.184 rows=74 loops=1)
|         -> Group aggregate: avg(a.Rating) (cost=280.65 rows=1392) (actual time=0.134..0.154 rows=96 loops=1)
|           -> Index scan on a using Neighborhood (cost=141.45 rows=1392) (actual time=0.120..0.1736 rows=1391 loops=1)
|
+-----+
```

The Cost remains 294.72-295.91, Time is 2.353-2.356 seconds

2. Index on Neighborhoods table for the n.AverageRent

```
mysql> CREATE INDEX idxAverageRent ON Neighborhoods (AverageRent)
```

After implementing these changes, we did not notice any change in the overall cost.

```
+-----+
| -> Limit: 15 row(s) (cost=294.72..295.91 rows=15) (actual time=2.656..2.659 rows=15 loops=1)
|   -> Table scan on <intersect temporary> (cost=294.72..297.56 rows=34) (actual time=2.655..2.658 rows=15 loops=1)
|     -> Intersect materialize with deduplication (cost=294.63..294.63 rows=34) (actual time=2.653..2.653 rows=48 loops=1)
|       -> Filter: (n.AverageRent < 2000) (cost=10.55 rows=34) (actual time=0.058..0.100 rows=48 loops=1)
|         -> Table scan on n (cost=10.55 rows=103) (actual time=0.056..0.090 rows=103 loops=1)
|       -> Filter: (avg(a.Rating) > 5) (cost=280.65 rows=1392) (actual time=0.267..2.458 rows=74 loops=1)
|         -> Group aggregate: avg(a.Rating) (cost=280.65 rows=1392) (actual time=0.259..2.426 rows=96 loops=1)
|           -> Index scan on a using Neighborhood (cost=141.45 rows=1392) (actual time=0.241..2.017 rows=1391 loops=1)
|
| +-----+
```

The Cost remains 294.72-295.91, Time is 2.656-2.659 seconds

3. Index on Apartments table for the Rating

```
mysql> CREATE INDEX idxRating ON Apartments (Rating)
```

After implementing these changes, we did not notice any change in the overall cost.

```

| -> Limit: 15 row(s) (cost=294.72..295.91 rows=15) (actual time=2.580..2.583 rows=15 loops=1)
    -> Table scan on <intersect temporary> (cost=294.72..297.56 rows=34) (actual time=2.579..2.581 rows=15 loops=1)
        -> Intersect materialize with deduplication (cost=294.63..294.63 rows=34) (actual time=2.576..2.576 rows=48 loops=1)
            -> Filter: (n.AverageRent < 2000) (cost=10.55 rows=34) (actual time=0.083..0.108 rows=48 loops=1)
                -> Table scan on n (cost=10.55 rows=103) (actual time=0.081..0.098 rows=103 loops=1)
            -> Filter: (avg(a.Rating) > 5) (cost=280.65 rows=1392) (actual time=0.178..2.355 rows=74 loops=1)
                -> Group aggregate: avg(a.Rating) (cost=280.65 rows=1392) (actual time=0.171..2.324 rows=96 loops=1)
                    -> Index scan on a using Neighborhood (cost=141.45 rows=1392) (actual time=0.155..1.882 rows=1391 loops=1)
|

```

The Cost remains 294.72-295.91, Time is 2.580-2.583 seconds

Final Index Design would be selecting the original query as the indexing did not make any significant differences in changing the original query.

Advanced Query 4

Using this query, we will try to improve the costs with indexing.

Explain analyze

```

SELECT DISTINCT n.NeighborhoodName
FROM Prefers p
JOIN Users u ON p.ID = u.UserId
JOIN Neighborhoods n ON p.Interest = n.NeighborhoodName
WHERE n.AverageAge < 50
AND n.NeighborhoodName IN
    (SELECT pref.Interest
     FROM Prefers pref
     JOIN Users us ON pref.ID = us.UserId
     WHERE us.Gender = 'Male'
     GROUP BY pref.Interest
     HAVING COUNT(*) >
        (SELECT COUNT(*)
         FROM Prefers pref2
         JOIN Users us2 ON pref2.ID = us2.UserId
         WHERE pref2.Interest = pref.Interest AND us2.Gender = 'Female'))
LIMIT 15;

```

We first use Explain Analyze to find the performance of the original query.

```

-> Limit: 15 row(s) (cost=54.19..54.95 rows=15) (actual time=0.888..0.890 rows=12 loops=1)
-> Table scan on <temporary> (cost=54.19..57.37 rows=60) (actual time=0.887..0.889 rows=12 loops=1)
-> Temporary table with deduplication (cost=54.13..54.13 rows=60) (actual time=0.886..0.886 rows=12 loops=1)
-> Limit table size: 15 unique row(s)
-> Nested loop inner join (cost=48.16 rows=60) (actual time=0.715..0.869 rows=22 loops=1)
-> Nested loop inner join (cost=27.23 rows=60) (actual time=0.710..0.830 rows=22 loops=1)
-> Filter: ((n.AverageAge < 50) and <in_optimizer>(n.NeighborhoodName,n.NeighborhoodName in (select #2))) (cost=10.55 rows=34) (actual time=0.704..0.790 rows=12 loops=1)
-> Table scan on n (cost=10.55 rows=103) (actual time=0.044..0.061 rows=103 loops=1)
-> Select #2 (subquery in condition; run only once)
-> Filter: (n.NeighborhoodName = 'Materialized_subquery'.Interest)) (cost=48.57..48.57 rows=1) (actual time=0.011..0.011 rows=0 loops=61)
-> Limit: 1 row(s) (cost=48.47..48.47 rows=1) (actual time=0.011..0.011 rows=0 loops=61)
-> Index lookup on <materialized_subquery> using <auto distinct key> (Interest=n.NeighborhoodName) (actual time=0.011..0.011 rows=0 loops=61)
-> Materialize with deduplication (cost=48.47..48.47 rows=10) (actual time=0.643..0.643 rows=21 loops=1)
-> Filter: (count(0) > (select #3)) (cost=47.46 rows=10) (actual time=0.123..0.629 rows=21 loops=1)
-> Group aggregate: count(0) (cost=47.46 rows=10) (actual time=0.062..0.308 rows=35 loops=1)
-> Nested loop inner join (cost=46.45 rows=10) (actual time=0.053..0.288 rows=44 loops=1)
-> Covering index scan on pref using Interest (cost=11.10 rows=101) (actual time=0.030..0.050 rows=101 loops=1)
-> Filter: (us2.Gender = 'Male') (cost=0.25 rows=0.1) (actual time=0.002..0.002 rows=0 loops=101)
-> Single-row index lookup on us using PRIMARY (UserId=pref.ID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=101)
-> Select #3 (subquery in condition; dependent)
-> Aggregate: count(0) (cost=1.11 rows=1) (actual time=0.008..0.008 rows=1 loops=35)
-> Nested loop inner join (cost=1.10 rows=0.2) (actual time=0.006..0.008 rows=1 loops=35)
-> Covering index lookup on pref2 using Interest (Interest=pref.Interest) (cost=0.49 rows=2) (actual time=0.003..0.004 rows=2 loops=35)
-> Filter: (us2.Gender = 'Female') (cost=0.26 rows=0.1) (actual time=0.002..0.002 rows=0 loops=71)
-> Single-row index lookup on us2 using PRIMARY (UserId=pref2.ID) (cost=0.26 rows=1) (actual time=0.001..0.001 rows=1 loops=71)
-> Covering index lookup on p using Interest (Interest=n.NeighborhoodName) (cost=0.32 rows=2) (actual time=0.002..0.003 rows=2 loops=12)
-> Limit: 1 row(s) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=22)
-> Single-row covering index lookup on u using PRIMARY (UserId=p.ID) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=22)

```

This query has an overall cost of 54.19-54.95 and a time of 0.888-0.890 seconds

We then apply 3 indexing methods to try and attempt to improve our runtime:

4. Index on Users table for the UserId

```
mysql> CREATE INDEX idxUserId ON Users (UserId)
```

After implementing these changes, we noticed that the cost slightly increased.

```

-> Limit: 15 row(s) (cost=63.15..63.91 rows=15) (actual time=0.952..0.954 rows=12 loops=1)
-> Table scan on <temporary> (cost=63.15..66.34 rows=60) (actual time=0.951..0.953 rows=12 loops=1)
-> Temporary table with deduplication (cost=63.10..63.10 rows=60) (actual time=0.950..0.950 rows=12 loops=1)
-> Limit table size: 15 unique row(s)
-> Nested loop inner join (cost=57.12 rows=60) (actual time=0.773..0.933 rows=22 loops=1)
-> Nested loop inner join (cost=27.23 rows=60) (actual time=0.768..0.892 rows=22 loops=1)
-> Filter: ((n.AverageAge < 50) and <in_optimizer>(n.NeighborhoodName,n.NeighborhoodName in (select #2))) (cost=10.55 rows=34) (actual time=0.760..0.851 rows=12 loops=1)
-> Select #2 (subquery in condition; run only once)
-> Filter: (n.NeighborhoodName = 'Materialized_subquery'.Interest)) (cost=63.72..63.72 rows=1) (actual time=0.012..0.012 rows=0 loops=61)
-> Limit: 1 row(s) (cost=63.62..63.62 rows=1) (actual time=0.012..0.012 rows=0 loops=61)
-> Index lookup on <materialized_subquery> using <auto distinct key> (Interest=n.NeighborhoodName) (actual time=0.012..0.012 rows=0 loops=61)
-> Materialize with deduplication (cost=63.62..63.62 rows=10) (actual time=0.681..0.681 rows=21 loops=1)
-> Filter: (count(0) > (select #3)) (cost=62.61 rows=10) (actual time=0.109..0.662 rows=21 loops=1)
-> Group aggregate: count(0) (cost=62.61 rows=10) (actual time=0.066..0.298 rows=35 loops=1)
-> Nested loop inner join (cost=61.60 rows=10) (actual time=0.057..0.277 rows=44 loops=1)
-> Covering index scan on pref using Interest (cost=11.10 rows=101) (actual time=0.028..0.050 rows=101 loops=1)
-> Filter: (us.Gender = 'Male') (cost=0.40 rows=0.1) (actual time=0.002..0.002 rows=0 loops=101)
-> Single-row index lookup on us using PRIMARY (UserId=pref.ID) (cost=0.40 rows=1) (actual time=0.002..0.002 rows=1 loops=101)
-> Select #3 (subquery in condition; dependent)
-> Aggregate: count(0) (cost=1.37 rows=1) (actual time=0.010..0.010 rows=1 loops=35)
-> Nested loop inner join (cost=1.36 rows=0.2) (actual time=0.007..0.009 rows=1 loops=35)
-> Covering index lookup on pref2 using Interest (Interest=pref.Interest) (cost=0.49 rows=2) (actual time=0.004..0.004 rows=2 loops=35)
-> Filter: (us2.Gender = 'Female') (cost=0.41 rows=0.1) (actual time=0.002..0.002 rows=0 loops=71)
-> Single-row index lookup on us2 using PRIMARY (UserId=pref2.ID) (cost=0.41 rows=1) (actual time=0.002..0.002 rows=1 loops=71)
-> Covering index lookup on p using Interest (Interest=n.NeighborhoodName) (cost=0.32 rows=2) (actual time=0.002..0.003 rows=2 loops=12)
-> Limit: 1 row(s) (cost=0.40 rows=1) (actual time=0.002..0.002 rows=1 loops=22)
-> Single-row covering index lookup on u using PRIMARY (UserId=p.ID) (cost=0.40 rows=1) (actual time=0.001..0.001 rows=1 loops=22)

```

The Cost increased to 63.15-63.91 and a time of 0.952-0.954 seconds

5. Index on Prefers table for the us.UserId

```
mysql> CREATE INDEX idxUserId ON Prefers (UserId)
```

After implementing these changes, we noticed that the cost was the same.

```

1 -> Limit: 15 row(s) (cost=54.19..54.95 rows=15) (actual time=0.989..0.991 rows=12 loops=1)
   -> Table scan on <temporary> (cost=54.19..57.37 rows=60) (actual time=0.989..0.990 rows=12 loops=1)
       -> Temporary table with deduplication (cost=54.13..54.13 rows=60) (actual time=0.987..0.987 rows=12 loops=1)
           -> Limit table size: 15 unique row(s)
               -> Nested loop inner join (cost=48.16 rows=60) (actual time=0.806..0.966 rows=22 loops=1)
                   -> Filter: ((n.AverageAge < 50) and <in_optimizer>(n.NeighborhoodName,n.NeighborhoodName in (select #2))) (cost=10.55 rows=34) (actual time=0.793..0.885 rows=12 loops=1)
                       -> Table scan on n (cost=10.55 rows=103) (actual time=0.064..0.087 rows=103 loops=1)
                           -> Select #2 (subquery in condition; run only once)
                               -> Filter: ((n.NeighborhoodName = <materialized subquery>'.Interest')) (cost=48.57..48.57 rows=1) (actual time=0.012..0.012 rows=0 loops=61)
                                   -> Limit: 1 row(s) (cost=48.47..48.47 rows=1) (actual time=0.012..0.012 rows=0 loops=61)
                                       -> Index lookup on <materialized subquery> using <auto distinct key> (Interest=n.NeighborhoodName) (actual time=0.012..0.012 rows=0 loops=61)
                                           -> Materialize with deduplication (cost=48.47..48.47 rows=10) (actual time=0.706..0.706 rows=21 loops=1)
                                               -> Filter: (count(0) > (select #3)) (cost=47.46 rows=10) (actual time=0.125..0.686 rows=21 loops=1)
                                                   -> Group aggregate: count(0) (cost=47.46 rows=10) (actual time=0.080..0.330 rows=35 loops=1)
                                                       -> Nested loop inner join (cost=46.45 rows=10) (actual time=0.070..0.201 rows=44 loops=1)
                                                           -> Covering index scan on pref using Interest (cost=11.10 rows=101) (actual time=0.037..0.062 rows=101 loops=1)
                                                           -> Filter: (us2.Gender = 'Male') (cost=0.25 rows=0.1) (actual time=0.002..0.002 rows=0 loops=101)
                                                               -> Single-row index lookup on us using PRIMARY (UserId=pref.ID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=101)
                                                                   -> Aggregate: count(0) (cost=1.11 rows=1) (actual time=0.009..0.009 rows=1 loops=35)
                                                                       -> Nested loop inner join (cost=1.10 rows=0.2) (actual time=0.007..0.009 rows=1 loops=35)
                                                                           -> Filter: (us2.Gender = 'Female') (cost=0.26 rows=0.1) (actual time=0.002..0.002 rows=0 loops=71)
                                                                               -> Single-row index lookup on us2 using PRIMARY (UserId=pref2.ID) (cost=0.26 rows=1) (actual time=0.001..0.002 rows=1 loops=71)
                                                                                   -> Select #3 (subquery in projection; dependent)
                                                                                       -> Aggregate: count(0) (cost=1.11 rows=1) (actual time=0.009..0.009 rows=1 loops=35)
                                                                                           -> Nested loop inner join (cost=1.10 rows=0.2) (actual time=0.007..0.009 rows=1 loops=35)
                                                                                               -> Covering index lookup on pref2 using Interest (Interest=pref.Interest) (cost=0.49 rows=2) (actual time=0.003..0.004 rows=2 loops=35)
                                                                                                   -> Filter: (us2.Gender = 'Female') (cost=0.26 rows=0.1) (actual time=0.002..0.002 rows=0 loops=71)
                                                                                                       -> Single-row index lookup on us2 using PRIMARY (UserId=pref2.ID) (cost=0.26 rows=1) (actual time=0.001..0.002 rows=1 loops=71)
                                                                                                           -> Covering index lookup on p using Interest (Interest=n.NeighborhoodName) (cost=0.32 rows=2) (actual time=0.002..0.003 rows=2 loops=12)
                                                                                                               -> Limit: 1 row(s) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=22)
                                                                                                                   -> Single-row covering index lookup on u using PRIMARY (UserId=p.ID) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=22)

```

The Cost remained at 54.19-54.95 and a time of 0.989-0.991 seconds

6. Index on Prefers table for the Interest

mysql> CREATE INDEX idxInterest ON Prefers(Interest)

After implementing these changes, we did not notice any change in the overall cost.

```

1 -> Limit: 15 row(s) (cost=54.19..54.95 rows=15) (actual time=0.994..0.997 rows=12 loops=1)
   -> Table scan on <temporary> (cost=54.19..57.37 rows=60) (actual time=0.994..0.996 rows=12 loops=1)
       -> Temporary table with deduplication (cost=54.13..54.13 rows=60) (actual time=0.992..0.992 rows=12 loops=1)
           -> Limit table size: 15 unique row(s)
               -> Nested loop inner join (cost=48.16 rows=60) (actual time=0.806..0.971 rows=22 loops=1)
                   -> Filter: ((n.AverageAge < 50) and <in_optimizer>(n.NeighborhoodName,n.NeighborhoodName in (select #2))) (cost=10.55 rows=34) (actual time=0.793..0.883 rows=12 loops=1)
                       -> Table scan on n (cost=10.55 rows=103) (actual time=0.067..0.089 rows=103 loops=1)
                           -> Select #2 (subquery in condition; run only once)
                               -> Filter: ((n.NeighborhoodName = <materialized subquery>'.Interest')) (cost=47.82..47.82 rows=1) (actual time=0.012..0.012 rows=0 loops=61)
                                   -> Limit: 1 row(s) (cost=47.72..47.72 rows=1) (actual time=0.012..0.012 rows=0 loops=61)
                                       -> Index lookup on <materialized subquery> using <auto distinct key> (Interest=n.NeighborhoodName) (actual time=0.012..0.012 rows=0 loops=61)
                                           -> Materialize with deduplication (cost=47.72..47.72 rows=10) (actual time=0.705..0.705 rows=21 loops=1)
                                               -> Filter: (count(0) > (select #3)) (cost=46.71 rows=10) (actual time=0.125..0.685 rows=21 loops=1)
                                                   -> Group aggregate: count(0) (cost=46.71 rows=10) (actual time=0.079..0.319 rows=35 loops=1)
                                                       -> Nested loop inner join (cost=45.70 rows=10) (actual time=0.069..0.237 rows=44 loops=1)
                                                           -> Covering index scan on pref using idxInterest (cost=10.35 rows=101) (actual time=0.036..0.058 rows=101 loops=1)
                                                           -> Filter: (us2.Gender = 'Male') (cost=0.25 rows=0.1) (actual time=0.002..0.002 rows=0 loops=101)
                                                               -> Single-row index lookup on us using PRIMARY (UserId=pref.ID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=101)
                                                                   -> Select #3 (subquery in condition; dependent)
                                                                                   -> Aggregate: count(0) (cost=1.11 rows=1) (actual time=0.010..0.010 rows=1 loops=35)
                                                                                       -> Nested loop inner join (cost=1.10 rows=0.2) (actual time=0.006..0.008 rows=1 loops=35)
                                                                                           -> Covering index lookup on pref2 using idxInterest (Interest=pref.Interest) (cost=0.49 rows=2) (actual time=0.003..0.004 rows=2 loops=35)
                                                                                               -> Filter: (us2.Gender = 'Female') (cost=0.26 rows=0.1) (actual time=0.002..0.002 rows=0 loops=71)
                                                                                                   -> Single-row index lookup on us2 using PRIMARY (UserId=pref2.ID) (cost=0.26 rows=1) (actual time=0.001..0.001 rows=1 loops=71)
                                                                                                       -> Select #3 (subquery in projection; dependent)
                                                                                                           -> Aggregate: count(0) (cost=1.11 rows=1) (actual time=0.010..0.010 rows=1 loops=35)
                                                                                                               -> Nested loop inner join (cost=1.10 rows=0.2) (actual time=0.006..0.008 rows=1 loops=35)
                                                                                                                   -> Covering index lookup on pref2 using idxInterest (Interest=pref.Interest) (cost=0.49 rows=2) (actual time=0.003..0.004 rows=2 loops=35)
                                                                                                                       -> Filter: (us2.Gender = 'Female') (cost=0.26 rows=0.1) (actual time=0.002..0.002 rows=0 loops=71)
                                                                                                                           -> Single-row index lookup on us2 using PRIMARY (UserId=pref2.ID) (cost=0.26 rows=1) (actual time=0.001..0.001 rows=1 loops=71)
                                                                                                                               -> Covering index lookup on p using idxInterest (Interest=n.NeighborhoodName) (cost=0.32 rows=2) (actual time=0.003..0.004 rows=2 loops=12)
                                                                                                                                   -> Limit: 1 row(s) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=22)
                                                                                                                                       -> Single-row covering index lookup on u using PRIMARY (UserId=p.ID) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=22)

```

The Cost remained at 54.19-54.95 and a time of 0.994-0.997 seconds

Final Index Design would be selecting the original query as the indexing did not make any significant differences in changing the original query. There was one change that increased the cost of the query so we will be avoiding that index.