SuiteMate

No More Gloomy Roomie :)

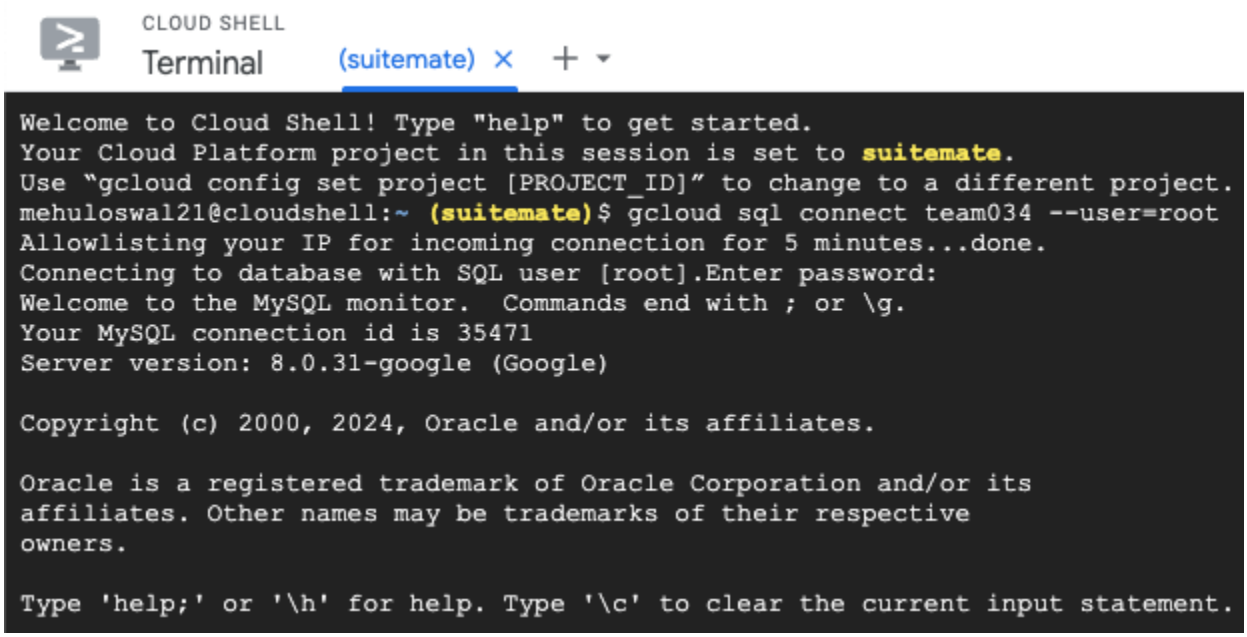
Project Track 1: Stage 3

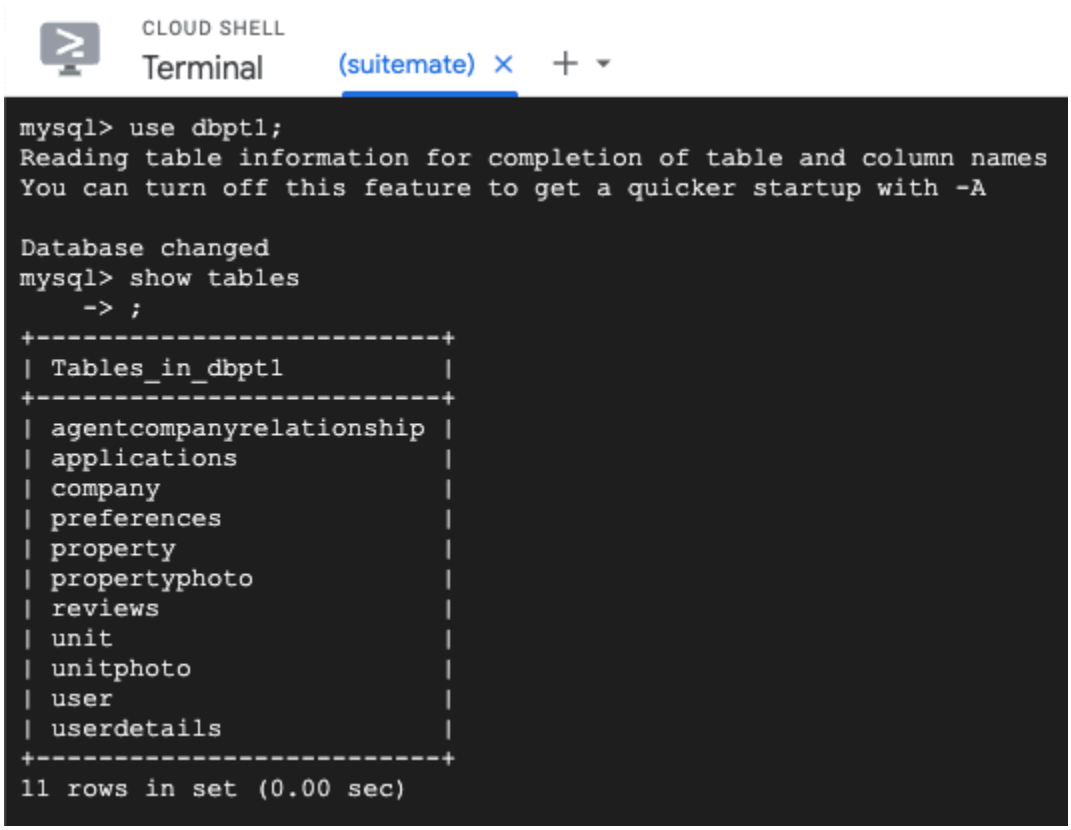
Database Design

Team034 - TeamNescafe

Mehul Oswal - mehuljo2
Daksh Gandhi - dakshg3
Daksh Pokar - dakshp2
Dhruv Kathpalia - dhruvk5

GCP Connection and Tables





DDL Commands

User:

```
CREATE TABLE User (
user_id INT PRIMARY KEY,
email_id VARCHAR(255) UNIQUE,
password_hash VARCHAR(255),
role_type ENUM('Customer', 'Agent'),
first_name VARCHAR(255),
last_name VARCHAR(255),
phone_number VARCHAR(20),
gender VARCHAR(10),
date_of_birth DATE
);
```

Company:

```
CREATE TABLE Company(
company_id INT PRIMARY KEY,
name VARCHAR(255),
phone_number VARCHAR(20),
hq_address VARCHAR(255),
website VARCHAR(255)
);
```

Property:

```
CREATE TABLE Property(
property_id INT PRIMARY KEY,
name VARCHAR(255),
address VARCHAR(255),
latitude REAL,
longitude REAL,
company_id INT,
pincode INT,
FOREIGN KEY (company_id) REFERENCES Company(company_id) ON UPDATE
CASCADE ON DELETE CASCADE
);
```

Unit:

```
CREATE TABLE Unit(
unit_id INT PRIMARY KEY,
property_id INT,
apartment_no INT,
bedrooms INT,
bathrooms REAL,
price REAL,
availability BOOLEAN,
area INT,
FOREIGN KEY (property_id) REFERENCES Property(property_id) ON UPDATE CASCADE
ON DELETE CASCADE
);
```

Preferences:

```
CREATE TABLE Preferences(
pref_id INT PRIMARY KEY,
pref_name VARCHAR(255)
);
```

PropertyPhoto:

```
CREATE TABLE PropertyPhoto(
property_id INT,
photo VARCHAR(255),
PRIMARY KEY(property_id, photo),
FOREIGN KEY (property_id) REFERENCES Property(property_id) ON UPDATE CASCADE
ON DELETE CASCADE
);
```

UnitPhoto:

```
CREATE TABLE UnitPhoto(
unit_id INT,
photo VARCHAR(255),
PRIMARY KEY(unit_id, photo),
FOREIGN KEY (unit_id) REFERENCES Unit(unit_id) ON UPDATE CASCADE ON DELETE
CASCADE
);
```

Applications:

```
CREATE TABLE Applications(
user_id INT,
unit_id INT,
created_at DATE,
status VARCHAR(255),
PRIMARY KEY (user_id, unit_id),
FOREIGN KEY (user_id) REFERENCES User(user_id) ON UPDATE CASCADE ON
DELETE CASCADE,
FOREIGN KEY (unit_id) REFERENCES Unit(unit_id) ON UPDATE CASCADE ON DELETE
CASCADE
);
```

Reviews:

```
CREATE TABLE Reviews(
user_id INT,
property_id INT,
created_at DATE,
comment VARCHAR(255),
rating INT,
PRIMARY KEY(user_id, property_id),
FOREIGN KEY (user_id) REFERENCES User(user_id) ON UPDATE CASCADE ON
DELETE CASCADE,
```

```
FOREIGN KEY (property_id) REFERENCES Property(property_id) ON UPDATE CASCADE
ON DELETE CASCADE
);
```

UserDetails:

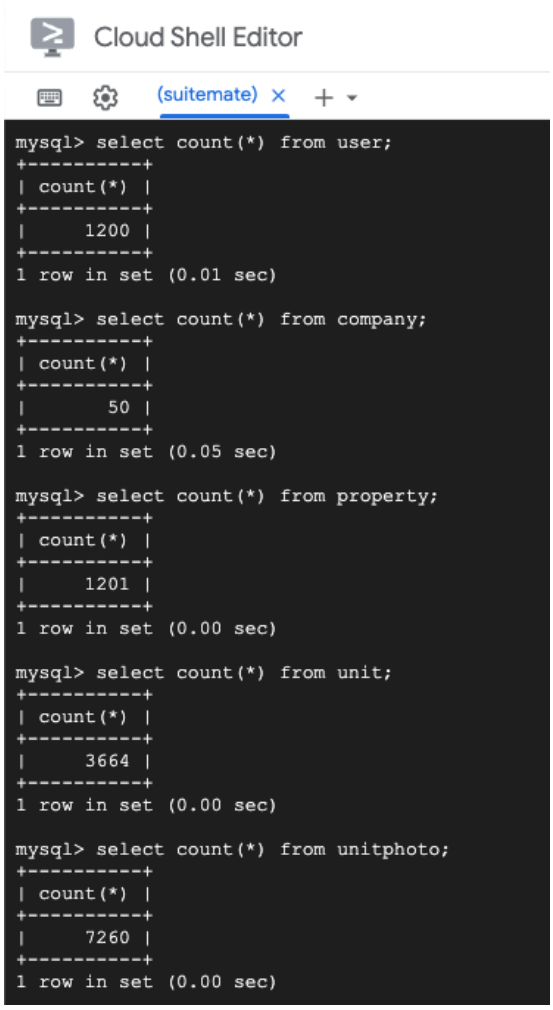
```
CREATE TABLE UserDetails(
user_id INT,
pref_id INT,
value VARCHAR(255),
PRIMARY KEY (user_id, pref_id),
FOREIGN KEY (user_id) REFERENCES User(user_id) ON UPDATE CASCADE ON
DELETE CASCADE,
FOREIGN KEY (pref_id) REFERENCES Preferences(pref_id) ON UPDATE CASCADE ON
DELETE CASCADE
);
```

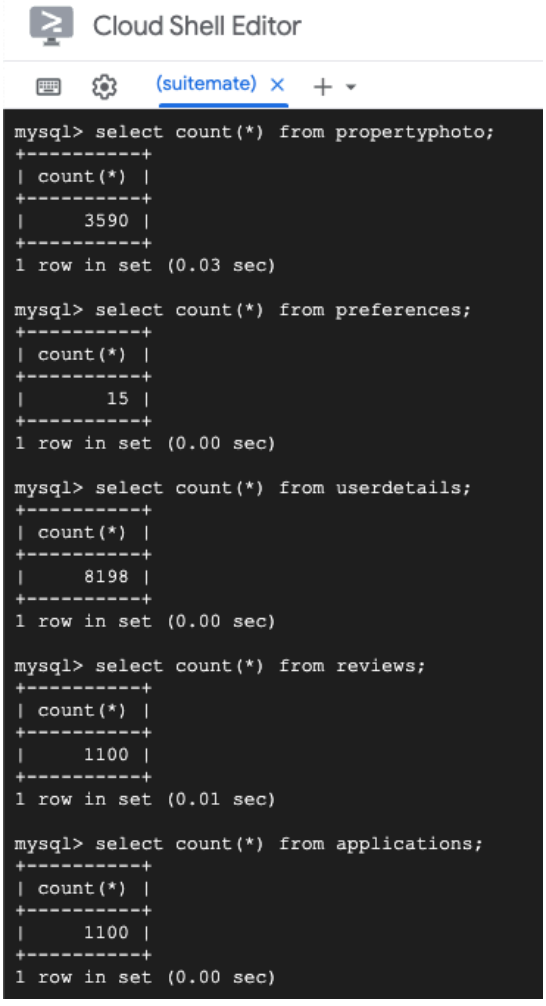
AgentCompanyRelationship:

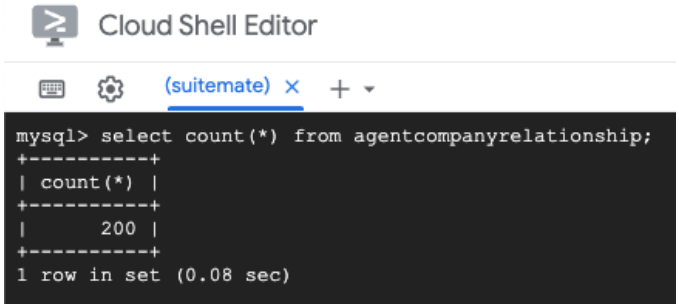
```
CREATE TABLE AgentCompanyRelationship(
user_id INT,
company_id INT,
PRIMARY KEY(user_id, company_id),
FOREIGN KEY (user_id) REFERENCES User(user_id) ON UPDATE CASCADE ON
DELETE CASCADE,
FOREIGN KEY (company_id) REFERENCES Company(company_id) ON UPDATE
CASCADE ON DELETE CASCADE
);
```

Count of Tables

Displaying the count of all 11 tables:





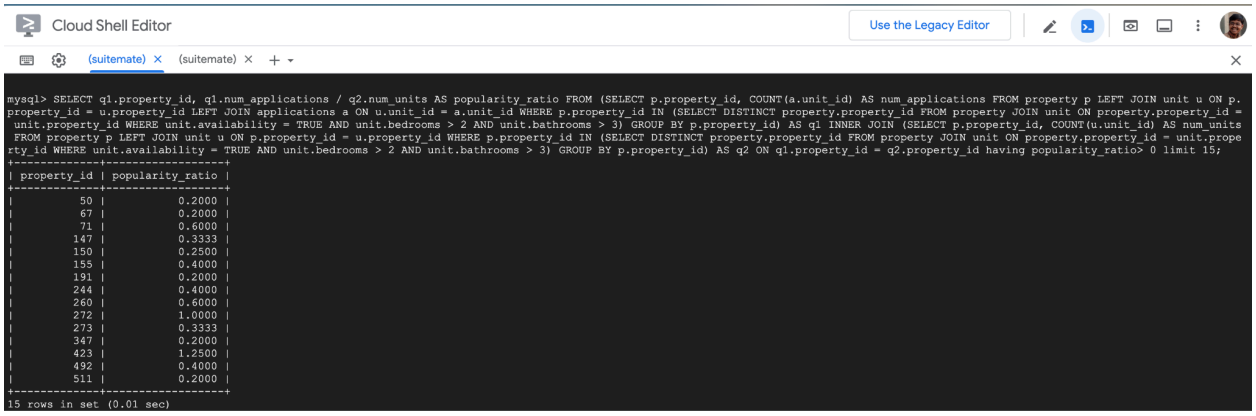


Complex Query 1:

Query

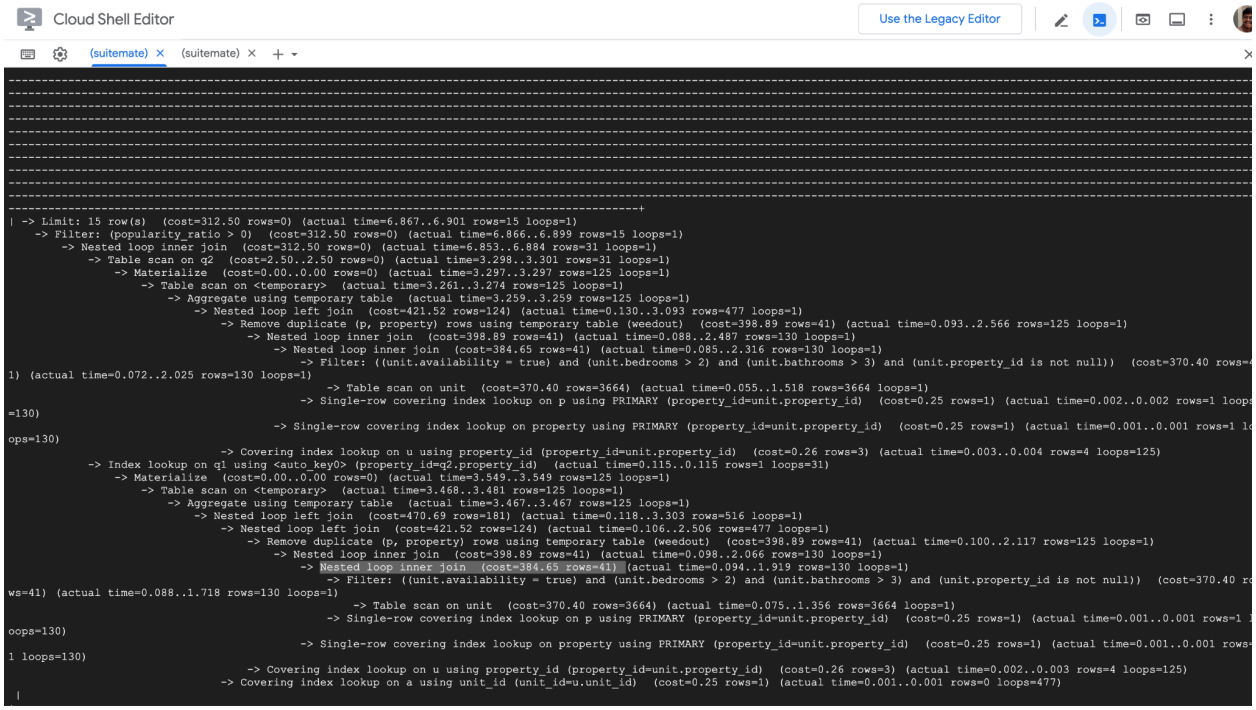
To determine the most popular properties based on a popularity ratio, calculated by dividing the number of applications per property by the number of units within each property.

```
SELECT q1.property_id, q1.num_applications / q2.num_units AS popularity_ratio FROM
(SELECT p.property_id, COUNT(a.unit_id) AS num_applications FROM property p LEFT
JOIN unit u ON p.property_id = u.property_id LEFT JOIN applications a ON u.unit_id =
a.unit_id WHERE p.property_id IN
(SELECT DISTINCT property.property_id FROM property JOIN unit ON property.property_id
= unit.property_id WHERE unit.availability = TRUE AND unit.bedrooms > 2 AND
unit.bathrooms > 2)
GROUP BY p.property_id) AS q1 INNER JOIN (SELECT p.property_id, COUNT(u.unit_id)
AS num_units FROM property p LEFT JOIN unit u ON p.property_id = u.property_id WHERE
p.property_id IN (SELECT DISTINCT property.property_id FROM property JOIN unit ON
property.property_id = unit.property_id WHERE unit.availability = TRUE AND unit.bedrooms >
2 AND unit.bathrooms > 2) GROUP BY p.property_id) AS q2 ON q1.property_id =
q2.property_id HAVING popularity_ratio> 0 LIMIT 15;
```



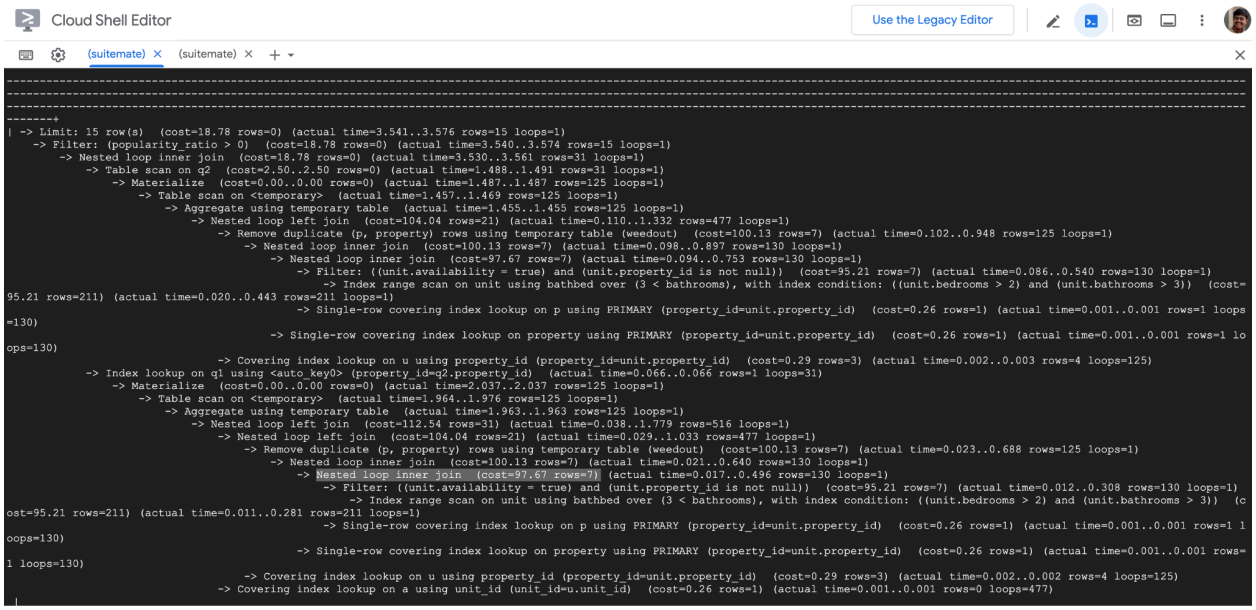

```
mysql> SELECT q1.property_id, q1.num_applications / q2.num_units AS popularity_ratio FROM (SELECT p.property_id, COUNT(a.unit_id) AS num_applications FROM property p LEFT JOIN unit u ON p.
property_id = u.property_id LEFT JOIN applications a ON u.unit_id = a.unit_id WHERE p.property_id IN (SELECT DISTINCT property.property_id FROM property JOIN unit ON property.property_id =
 unit.property_id WHERE unit.availability = TRUE AND unit.bedrooms > 2 AND unit.bathrooms > 3) GROUP BY p.property_id) AS q1 INNER JOIN (SELECT p.property_id, COUNT(u.unit_id) AS num_units
 FROM property p LEFT JOIN unit u ON p.property_id = u.property_id WHERE p.property_id IN (SELECT DISTINCT property.property_id FROM property JOIN unit ON property.property_id = unit.prope
rty_id WHERE unit.availability = TRUE AND unit.bedrooms > 2 AND unit.bathrooms > 3) GROUP BY p.property_id) AS q2 ON q1.property_id = q2.property_id having popularity_ratio> 0 limit 15;
+-------------+------------------+
| property_id | popularity_ratio |
+-------------+------------------+
|          50 |           0.2000 |
|          67 |           0.2000 |
|          71 |           0.6000 |
|         147 |           0.3333 |
|         150 |           0.2500 |
|         155 |           0.4000 |
|         191 |           0.2000 |
|         244 |           0.4000 |
|         260 |           0.6000 |
|         272 |           1.0000 |
|         273 |           0.3333 |
|         347 |           0.2000 |
|         423 |           1.2500 |
|         492 |           0.4000 |
|         511 |           0.2000 |
+-------------+------------------+
15 rows in set (0.01 sec)
```

Default Index:



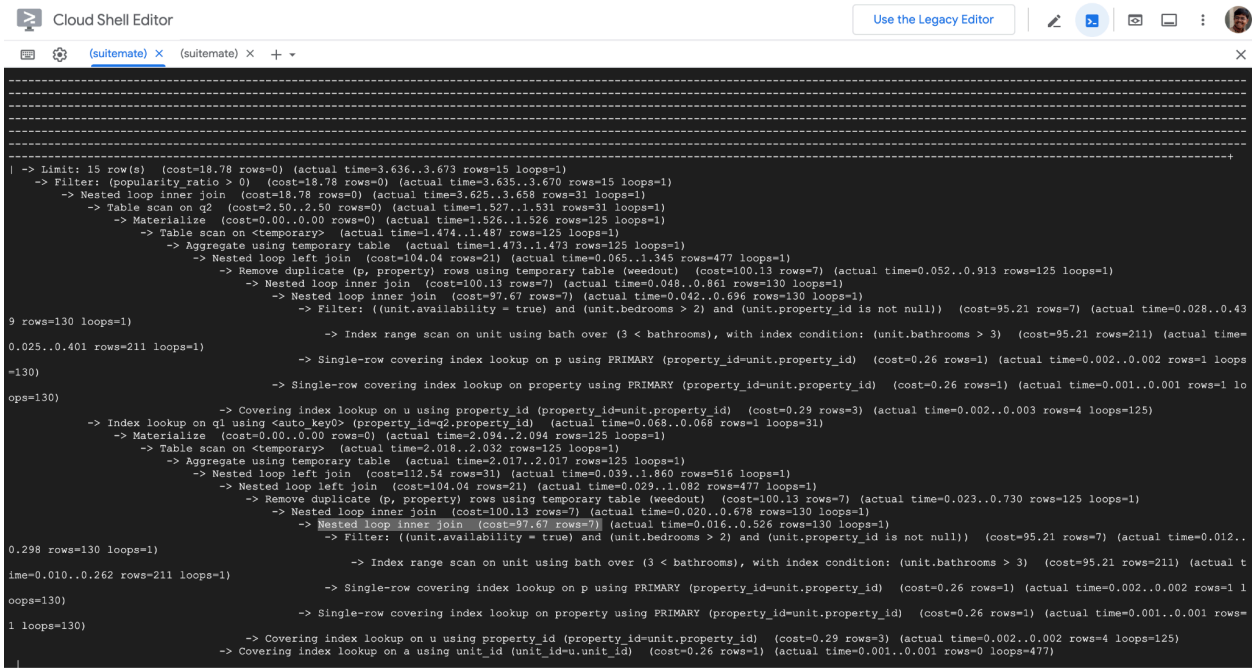

```
| -> Limit: 15 row(s)  (cost=312.50 rows=0) (actual time=6.867..6.901 rows=15 loops=1)
    -> Filter: (popularity_ratio > 0)  (cost=312.50 rows=0) (actual time=6.866..6.899 rows=15 loops=1)
        -> Nested loop inner join  (cost=312.50 rows=0) (actual time=6.853..6.884 rows=31 loops=1)
            -> Table scan on q2  (cost=2.50..2.50 rows=0) (actual time=3.298..3.301 rows=31 loops=1)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.297..3.297 rows=125 loops=1)
                    -> Table scan on <temporary>  (actual time=3.261..3.274 rows=125 loops=1)
                        -> Aggregate using temporary table  (actual time=3.259..3.259 rows=125 loops=1)
                            -> Nested loop left join  (cost=421.52 rows=124) (actual time=0.130..3.093 rows=477 loops=1)
                                -> Remove duplicate (p, property) rows using temporary table (weedout)  (cost=398.89 rows=41) (actual time=0.093..2.566 rows=125 loops=1)
                                    -> Nested loop inner join  (cost=398.89 rows=41) (actual time=0.088..2.487 rows=130 loops=1)
                                        -> Nested loop inner join  (cost=384.65 rows=41) (actual time=0.085..2.316 rows=130 loops=1)
                                            -> Filter: ((unit.availability = true) and (unit.bedrooms > 2) and (unit.bathrooms > 3) and (unit.property_id is not null))  (cost=370.40 rows=4
1) (actual time=0.072..2.025 rows=130 loops=1)
                                                -> Table scan on unit  (cost=370.40 rows=3664) (actual time=0.055..1.518 rows=3664 loops=1)
                                            -> Single-row covering index lookup on p using PRIMARY (property_id=unit.property_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops
=130)
                                        -> Single-row covering index lookup on property using PRIMARY (property_id=unit.property_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 lo
ops=130)
                                    -> Covering index lookup on u using property_id (property_id=unit.property_id)  (cost=0.26 rows=3) (actual time=0.003..0.004 rows=4 loops=125)
            -> Index lookup on q1 using <auto_key0> (property_id=q2.property_id)  (actual time=0.115..0.115 rows=1 loops=31)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.549..3.549 rows=125 loops=1)
                    -> Table scan on <temporary>  (actual time=3.468..3.481 rows=125 loops=1)
                        -> Aggregate using temporary table  (actual time=3.467..3.467 rows=125 loops=1)
                            -> Nested loop left join  (cost=470.69 rows=181) (actual time=0.118..3.303 rows=516 loops=1)
                                -> Nested loop left join  (cost=421.52 rows=124) (actual time=0.106..2.506 rows=477 loops=1)
                                    -> Remove duplicate (p, property) rows using temporary table (weedout)  (cost=398.89 rows=41) (actual time=0.100..2.117 rows=125 loops=1)
                                        -> Nested loop inner join  (cost=398.89 rows=41) (actual time=0.098..2.066 rows=130 loops=1)
                                            -> Nested loop inner join  (cost=384.65 rows=41) (actual time=0.094..1.919 rows=130 loops=1)
                                                -> Filter: ((unit.availability = true) and (unit.bedrooms > 2) and (unit.bathrooms > 3) and (unit.property_id is not null))  (cost=370.40 ro
ws=41) (actual time=0.088..1.718 rows=130 loops=1)
                                                    -> Table scan on unit  (cost=370.40 rows=3664) (actual time=0.075..1.356 rows=3664 loops=1)
                                            -> Single-row covering index lookup on p using PRIMARY (property_id=unit.property_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 l
oops=130)
                                        -> Single-row covering index lookup on property using PRIMARY (property_id=unit.property_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=
1 loops=130)
                                    -> Covering index lookup on u using property_id (property_id=unit.property_id)  (cost=0.26 rows=3) (actual time=0.002..0.003 rows=4 loops=125)
                                -> Covering index lookup on a using unit_id (unit_id=u.unit_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=0 loops=477)
 |
```

Index on (Bathroom, Bedroom):



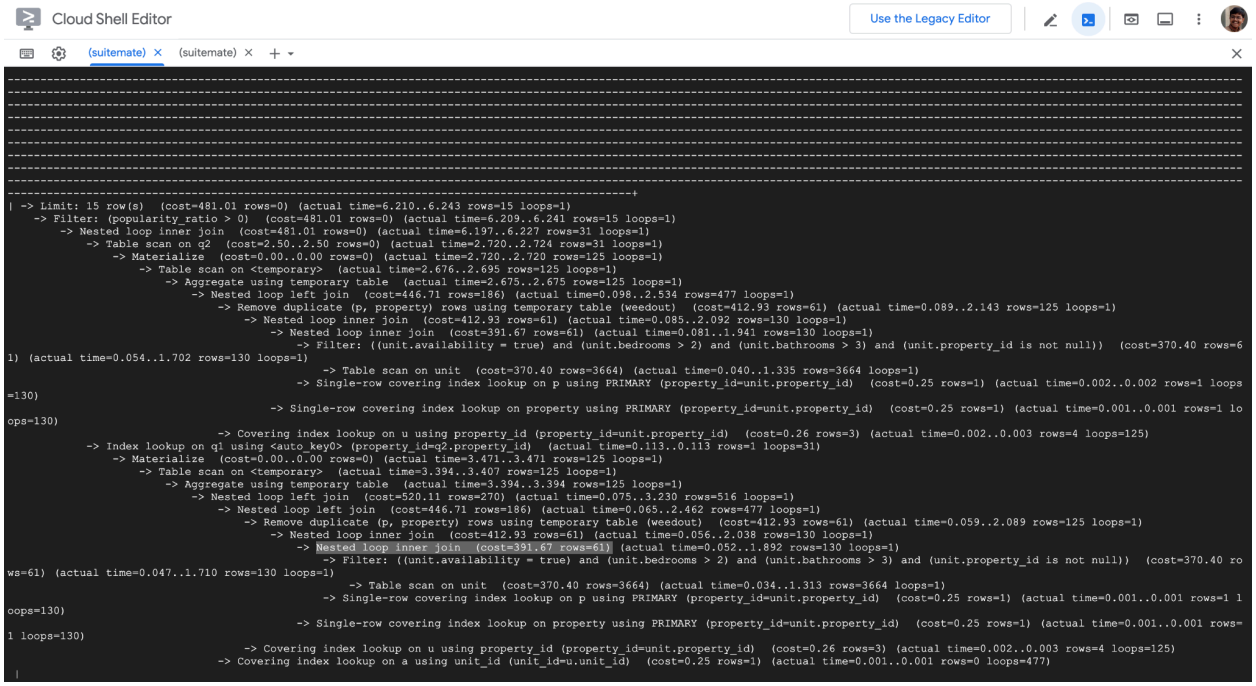

```
| -> Limit: 15 row(s)  (cost=18.78 rows=0) (actual time=3.541..3.576 rows=15 loops=1)
    -> Filter: (popularity_ratio > 0)  (cost=18.78 rows=0) (actual time=3.540..3.574 rows=15 loops=1)
        -> Nested loop inner join  (cost=18.78 rows=0) (actual time=3.530..3.561 rows=31 loops=1)
            -> Table scan on q2  (cost=2.50..2.50 rows=0) (actual time=1.488..1.491 rows=31 loops=1)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1.487..1.487 rows=125 loops=1)
                    -> Table scan on <temporary>  (actual time=1.457..1.469 rows=125 loops=1)
                        -> Aggregate using temporary table  (actual time=1.455..1.455 rows=125 loops=1)
                            -> Nested loop left join  (cost=104.04 rows=21) (actual time=0.110..1.332 rows=477 loops=1)
                                -> Remove duplicate (p, property) rows using temporary table (weedout)  (cost=100.13 rows=7) (actual time=0.102..0.948 rows=125 loops=1)
                                    -> Nested loop inner join  (cost=100.13 rows=7) (actual time=0.098..0.897 rows=130 loops=1)
                                        -> Nested loop inner join  (cost=97.67 rows=7) (actual time=0.094..0.753 rows=130 loops=1)
                                            -> Filter: ((unit.availability = true) and (unit.property_id is not null))  (cost=95.21 rows=7) (actual time=0.086..0.540 rows=130 loops=1)
                                                -> Index range scan on unit using bathbed over (3 < bathrooms), with index condition: ((unit.bedrooms > 2) and (unit.bathrooms > 3))  (cost=
95.21 rows=211) (actual time=0.020..0.443 rows=211 loops=1)
                                            -> Single-row covering index lookup on p using PRIMARY (property_id=unit.property_id)  (cost=0.26 rows=1) (actual time=0.001..0.001 rows=1 loops
=130)
                                        -> Single-row covering index lookup on property using PRIMARY (property_id=unit.property_id)  (cost=0.26 rows=1) (actual time=0.001..0.001 rows=1 lo
ops=130)
                                    -> Covering index lookup on u using property_id (property_id=unit.property_id)  (cost=0.29 rows=3) (actual time=0.002..0.003 rows=4 loops=125)
            -> Index lookup on q1 using <auto_key0> (property_id=q2.property_id)  (actual time=0.066..0.066 rows=1 loops=31)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.037..2.037 rows=125 loops=1)
                    -> Table scan on <temporary>  (actual time=1.964..1.976 rows=125 loops=1)
                        -> Aggregate using temporary table  (actual time=1.963..1.963 rows=125 loops=1)
                            -> Nested loop left join  (cost=112.54 rows=31) (actual time=0.038..1.779 rows=516 loops=1)
                                -> Nested loop left join  (cost=104.04 rows=21) (actual time=0.029..1.033 rows=477 loops=1)
                                    -> Remove duplicate (p, property) rows using temporary table (weedout)  (cost=100.13 rows=7) (actual time=0.023..0.688 rows=125 loops=1)
                                        -> Nested loop inner join  (cost=100.13 rows=7) (actual time=0.021..0.640 rows=130 loops=1)
                                            -> Nested loop inner join  (cost=97.67 rows=7) (actual time=0.017..0.496 rows=130 loops=1)
                                                -> Filter: ((unit.availability = true) and (unit.property_id is not null))  (cost=95.21 rows=7) (actual time=0.012..0.308 rows=130 loops=1)
                                                    -> Index range scan on unit using bathbed over (3 < bathrooms), with index condition: ((unit.bedrooms > 2) and (unit.bathrooms > 3))  (c
ost=95.21 rows=211) (actual time=0.011..0.281 rows=211 loops=1)
                                            -> Single-row covering index lookup on p using PRIMARY (property_id=unit.property_id)  (cost=0.26 rows=1) (actual time=0.001..0.001 rows=1 l
oops=130)
                                        -> Single-row covering index lookup on property using PRIMARY (property_id=unit.property_id)  (cost=0.26 rows=1) (actual time=0.001..0.001 rows=
1 loops=130)
                                    -> Covering index lookup on u using property_id (property_id=unit.property_id)  (cost=0.29 rows=3) (actual time=0.002..0.002 rows=4 loops=125)
                                -> Covering index lookup on a using unit_id (unit_id=u.unit_id)  (cost=0.26 rows=1) (actual time=0.001..0.001 rows=0 loops=477)
 |
```

Index on Bathroom:



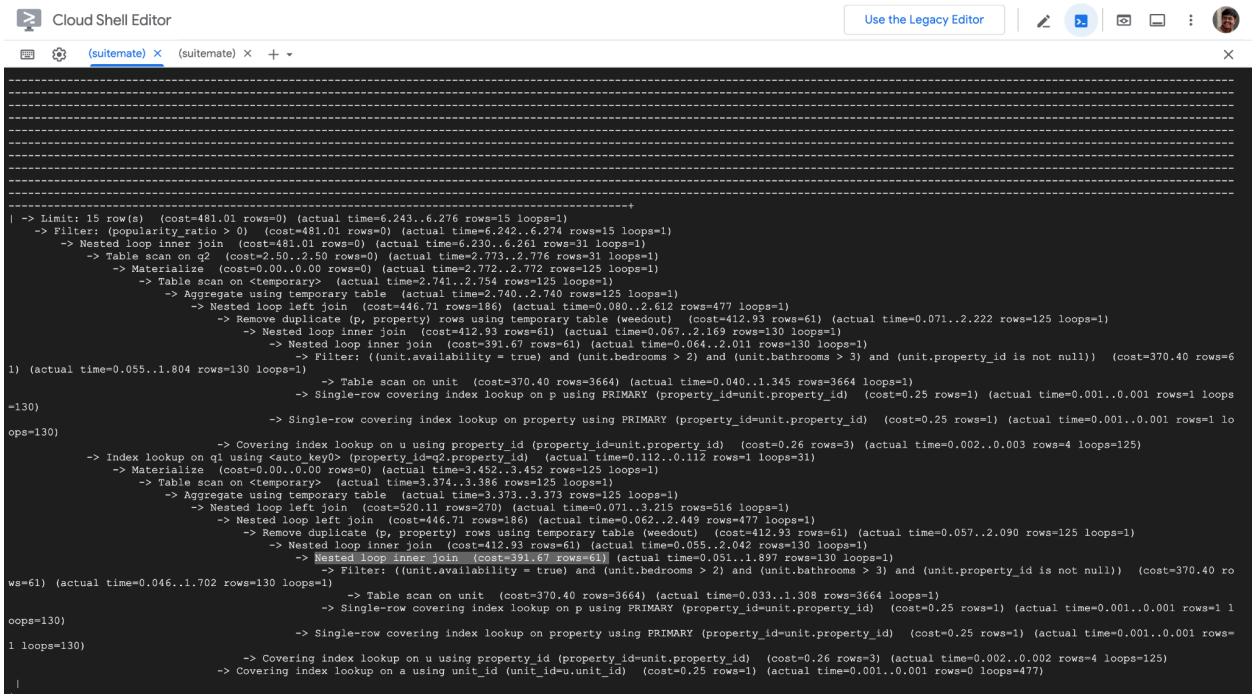

```
-> Limit: 15 row(s)  (cost=18.78 rows=0) (actual time=3.636..3.673 rows=15 loops=1)
    -> Filter: (popularity_ratio > 0)  (cost=18.78 rows=0) (actual time=3.635..3.670 rows=15 loops=1)
        -> Nested loop inner join  (cost=18.78 rows=0) (actual time=3.625..3.658 rows=31 loops=1)
            -> Table scan on q2  (cost=2.50..2.50 rows=0) (actual time=1.527..1.531 rows=31 loops=1)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1.526..1.526 rows=125 loops=1)
                    -> Table scan on <temporary>  (actual time=1.474..1.487 rows=125 loops=1)
                        -> Aggregate using temporary table  (actual time=1.473..1.473 rows=125 loops=1)
                            -> Nested loop left join  (cost=104.04 rows=21) (actual time=0.065..1.345 rows=477 loops=1)
                                -> Remove duplicate (p, property) rows using temporary table (weedout)  (cost=100.13 rows=7) (actual time=0.052..0.913 rows=125 loops=1)
                                    -> Nested loop inner join  (cost=100.13 rows=7) (actual time=0.048..0.861 rows=130 loops=1)
                                        -> Nested loop inner join  (cost=97.67 rows=7) (actual time=0.042..0.696 rows=130 loops=1)
                                            -> Filter: ((unit.availability = true) and (unit.bedrooms > 2) and (unit.property_id is not null))  (cost=95.21 rows=7) (actual time=0.028..0.43
9 rows=130 loops=1)
                                                -> Index range scan on unit using bath_over (3 < bathrooms), with index condition: (unit.bathrooms > 3)  (cost=95.21 rows=211) (actual time=
0.025..0.401 rows=211 loops=1)
                                            -> Single-row covering index lookup on p using PRIMARY (property_id=unit.property_id)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops
=130)
                                        -> Single-row covering index lookup on property using PRIMARY (property_id=unit.property_id)  (cost=0.26 rows=1) (actual time=0.001..0.001 rows=1 lo
ops=130)
                                -> Covering index lookup on u using property_id (property_id=unit.property_id)  (cost=0.29 rows=3) (actual time=0.002..0.003 rows=4 loops=125)
            -> Index lookup on q1 using <auto_key0> (property_id=q2.property_id)  (actual time=0.068..0.068 rows=1 loops=31)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.094..2.094 rows=125 loops=1)
                    -> Table scan on <temporary>  (actual time=2.018..2.032 rows=125 loops=1)
                        -> Aggregate using temporary table  (actual time=2.017..2.017 rows=125 loops=1)
                            -> Nested loop left join  (cost=112.54 rows=31) (actual time=0.039..1.860 rows=516 loops=1)
                                -> Nested loop left join  (cost=104.04 rows=21) (actual time=0.029..1.082 rows=477 loops=1)
                                    -> Remove duplicate (p, property) rows using temporary table (weedout)  (cost=100.13 rows=7) (actual time=0.023..0.730 rows=125 loops=1)
                                        -> Nested loop inner join  (cost=100.13 rows=7) (actual time=0.020..0.678 rows=130 loops=1)
                                            -> Nested loop inner join  (cost=97.67 rows=7) (actual time=0.016..0.526 rows=130 loops=1)
                                                -> Filter: ((unit.availability = true) and (unit.bedrooms > 2) and (unit.property_id is not null))  (cost=95.21 rows=7) (actual time=0.012..
0.298 rows=130 loops=1)
                                                    -> Index range scan on unit using bath_over (3 < bathrooms), with index condition: (unit.bathrooms > 3)  (cost=95.21 rows=211) (actual t
ime=0.010..0.262 rows=211 loops=1)
                                                -> Single-row covering index lookup on p using PRIMARY (property_id=unit.property_id)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 l
oops=130)
                                            -> Single-row covering index lookup on property using PRIMARY (property_id=unit.property_id)  (cost=0.26 rows=1) (actual time=0.001..0.001 rows=
1 loops=130)
                                    -> Covering index lookup on u using property_id (property_id=unit.property_id)  (cost=0.29 rows=3) (actual time=0.002..0.002 rows=4 loops=125)
                                -> Covering index lookup on a using unit_id (unit_id=u.unit_id)  (cost=0.26 rows=1) (actual time=0.001..0.001 rows=0 loops=477)
```

Index on (Bedroom, Availability):




```
-> Limit: 15 row(s)  (cost=481.01 rows=0) (actual time=6.210..6.243 rows=15 loops=1)
    -> Filter: (popularity_ratio > 0)  (cost=481.01 rows=0) (actual time=6.209..6.241 rows=15 loops=1)
        -> Nested loop inner join  (cost=481.01 rows=0) (actual time=6.197..6.227 rows=31 loops=1)
            -> Table scan on q2  (cost=2.50..2.50 rows=0) (actual time=2.720..2.724 rows=31 loops=1)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.720..2.720 rows=125 loops=1)
                    -> Table scan on <temporary>  (actual time=2.676..2.695 rows=125 loops=1)
                        -> Aggregate using temporary table  (actual time=2.675..2.675 rows=125 loops=1)
                            -> Nested loop left join  (cost=446.71 rows=186) (actual time=0.098..2.534 rows=477 loops=1)
                                -> Remove duplicate (p, property) rows using temporary table (weedout)  (cost=412.93 rows=61) (actual time=0.089..2.143 rows=125 loops=1)
                                    -> Nested loop inner join  (cost=412.93 rows=61) (actual time=0.085..2.092 rows=130 loops=1)
                                        -> Nested loop inner join  (cost=391.67 rows=61) (actual time=0.081..1.941 rows=130 loops=1)
                                            -> Filter: ((unit.availability = true) and (unit.bedrooms > 2) and (unit.bathrooms > 3) and (unit.property_id is not null))  (cost=370.40 rows=6
1) (actual time=0.054..1.702 rows=130 loops=1)
                                                -> Table scan on unit  (cost=370.40 rows=3664) (actual time=0.040..1.335 rows=3664 loops=1)
                                            -> Single-row covering index lookup on p using PRIMARY (property_id=unit.property_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops
=130)
                                        -> Single-row covering index lookup on property using PRIMARY (property_id=unit.property_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 lo
ops=130)
                                -> Covering index lookup on u using property_id (property_id=unit.property_id)  (cost=0.26 rows=3) (actual time=0.002..0.003 rows=4 loops=125)
            -> Index lookup on q1 using <auto_key0> (property_id=q2.property_id)  (actual time=0.113..0.113 rows=1 loops=31)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.471..3.471 rows=125 loops=1)
                    -> Table scan on <temporary>  (actual time=3.394..3.407 rows=125 loops=1)
                        -> Aggregate using temporary table  (actual time=3.394..3.394 rows=125 loops=1)
                            -> Nested loop left join  (cost=520.11 rows=270) (actual time=0.075..3.230 rows=516 loops=1)
                                -> Nested loop left join  (cost=446.71 rows=186) (actual time=0.065..2.462 rows=477 loops=1)
                                    -> Remove duplicate (p, property) rows using temporary table (weedout)  (cost=412.93 rows=61) (actual time=0.059..2.089 rows=125 loops=1)
                                        -> Nested loop inner join  (cost=412.93 rows=61) (actual time=0.056..2.038 rows=130 loops=1)
                                            -> Nested loop inner join  (cost=391.67 rows=61) (actual time=0.052..1.892 rows=130 loops=1)
                                                -> Filter: ((unit.availability = true) and (unit.bedrooms > 2) and (unit.bathrooms > 3) and (unit.property_id is not null))  (cost=370.40 ro
ws=61) (actual time=0.047..1.710 rows=130 loops=1)
                                                    -> Table scan on unit  (cost=370.40 rows=3664) (actual time=0.034..1.313 rows=3664 loops=1)
                                                -> Single-row covering index lookup on p using PRIMARY (property_id=unit.property_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 l
oops=130)
                                            -> Single-row covering index lookup on property using PRIMARY (property_id=unit.property_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=
1 loops=130)
                                    -> Covering index lookup on u using property_id (property_id=unit.property_id)  (cost=0.26 rows=3) (actual time=0.002..0.003 rows=4 loops=125)
                                -> Covering index lookup on a using unit_id (unit_id=u.unit_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=0 loops=477)
```

Index on Bedroom:




```
-> Limit: 15 row(s)  (cost=481.01 rows=0) (actual time=6.243..6.276 rows=15 loops=1)
    -> Filter: (popularity_ratio > 0)  (cost=481.01 rows=0) (actual time=6.242..6.274 rows=15 loops=1)
        -> Nested loop inner join  (cost=481.01 rows=0) (actual time=6.230..6.261 rows=31 loops=1)
            -> Table scan on q2  (cost=2.50..2.50 rows=0) (actual time=2.773..2.776 rows=31 loops=1)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.772..2.772 rows=125 loops=1)
                    -> Table scan on <temporary>  (actual time=2.741..2.754 rows=125 loops=1)
                        -> Aggregate using temporary table  (actual time=2.740..2.740 rows=125 loops=1)
                            -> Nested loop left join  (cost=446.71 rows=186) (actual time=0.080..2.612 rows=477 loops=1)
                                -> Remove duplicate (p, property) rows using temporary table (weedout)  (cost=412.93 rows=61) (actual time=0.071..2.222 rows=125 loops=1)
                                    -> Nested loop inner join  (cost=412.93 rows=61) (actual time=0.067..2.169 rows=130 loops=1)
                                        -> Nested loop inner join  (cost=391.67 rows=61) (actual time=0.064..2.011 rows=130 loops=1)
                                            -> Filter: ((unit.availability = true) and (unit.bedrooms > 2) and (unit.bathrooms > 3) and (unit.property_id is not null))  (cost=370.40 rows=6
1) (actual time=0.055..1.804 rows=130 loops=1)
                                                -> Table scan on unit  (cost=370.40 rows=3664) (actual time=0.040..1.345 rows=3664 loops=1)
                                            -> Single-row covering index lookup on p using PRIMARY (property_id=unit.property_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops
=130)
                                        -> Single-row covering index lookup on property using PRIMARY (property_id=unit.property_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 lo
ops=130)
                                -> Covering index lookup on u using property_id (property_id=unit.property_id)  (cost=0.26 rows=3) (actual time=0.002..0.003 rows=4 loops=125)
            -> Index lookup on q1 using <auto_key0> (property_id=q2.property_id)  (actual time=0.112..0.112 rows=1 loops=31)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.452..3.452 rows=125 loops=1)
                    -> Table scan on <temporary>  (actual time=3.374..3.386 rows=125 loops=1)
                        -> Aggregate using temporary table  (actual time=3.373..3.373 rows=125 loops=1)
                            -> Nested loop left join  (cost=520.11 rows=270) (actual time=0.071..3.215 rows=516 loops=1)
                                -> Nested loop left join  (cost=446.71 rows=186) (actual time=0.062..2.449 rows=477 loops=1)
                                    -> Remove duplicate (p, property) rows using temporary table (weedout)  (cost=412.93 rows=61) (actual time=0.057..2.090 rows=125 loops=1)
                                        -> Nested loop inner join  (cost=412.93 rows=61) (actual time=0.055..2.042 rows=130 loops=1)
                                            -> Nested loop inner join  (cost=391.67 rows=61) (actual time=0.051..1.897 rows=130 loops=1)
                                                -> Filter: ((unit.availability = true) and (unit.bedrooms > 2) and (unit.bathrooms > 3) and (unit.property_id is not null))  (cost=370.40 ro
ws=61) (actual time=0.046..1.702 rows=130 loops=1)
                                                    -> Table scan on unit  (cost=370.40 rows=3664) (actual time=0.033..1.308 rows=3664 loops=1)
                                                -> Single-row covering index lookup on p using PRIMARY (property_id=unit.property_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 l
oops=130)
                                            -> Single-row covering index lookup on property using PRIMARY (property_id=unit.property_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=
1 loops=130)
                                    -> Covering index lookup on u using property_id (property_id=unit.property_id)  (cost=0.26 rows=3) (actual time=0.002..0.002 rows=4 loops=125)
                                -> Covering index lookup on a using unit_id (unit_id=u.unit_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=0 loops=477)
```

Analysis

We have tried 4 different combinations of indexing for the above query.

1. Default Index: We first ran the query using the default index and saw that the query engine was doing a table scan using the primary keys as the default index for each table.
2. (Bathroom, Bedroom): After indexing using these two attributes, we could see a significant loss in the cost. Using EXPLAIN ANALYZE, we could see that instead of the table scan, the index range scan reduced the cost significantly.
3. (Bathroom): Upon indexing with the bathroom attribute, the cost was reduced as compared to the default index. It was fetching rows and the cost was similar to that of (Bathroom, Bedroom).
4. (Bedroom, Availability): Since these attributes also determined the fetching of the rows, we tried indexing them. It was noticed that there was a rise in cost, instead of a reduction. This was majorly due to the table scan done instead of an index scan by the query engine.
5. (Bedroom): Upon indexing with the bedroom attribute, we see a rise in the cost than the default indexing. We also noticed that a table scan is taking place instead of an index-ranged one.

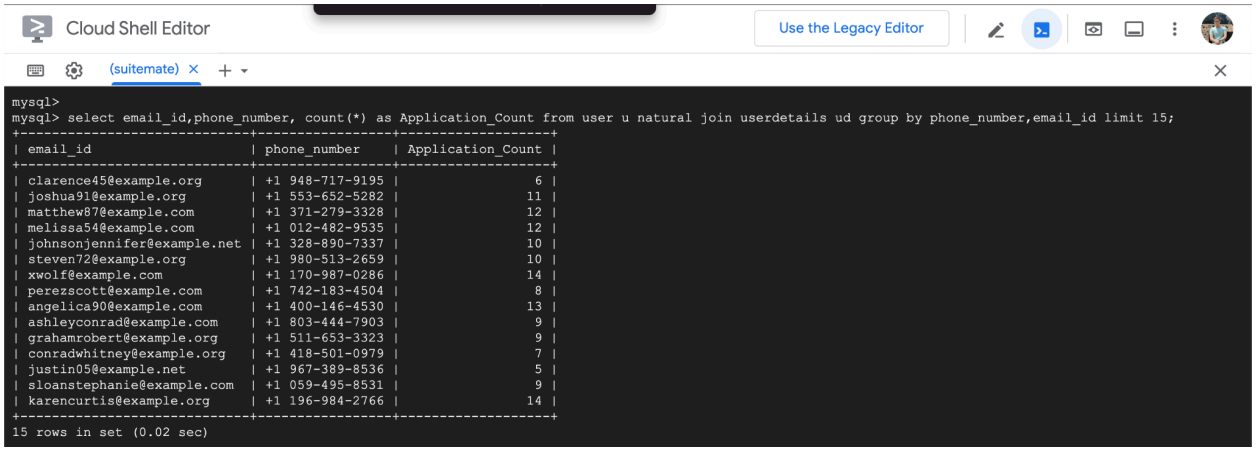
After the above combinations of indexing, indexing using the bathroom seems to be improving the overall cost as it has better cardinality in our data and thus fetches the rows optimally.

Complex Query 2:

Query

To generate analytics for leasing agents: Calculate the application count per user, grouping by phone number and email ID.

```
SELECT email_id,phone_number, count(*) AS Application_Count FROM user u NATURAL JOIN userdetails ud GROUP BY phone_number,email_id LIMIT 15;
```



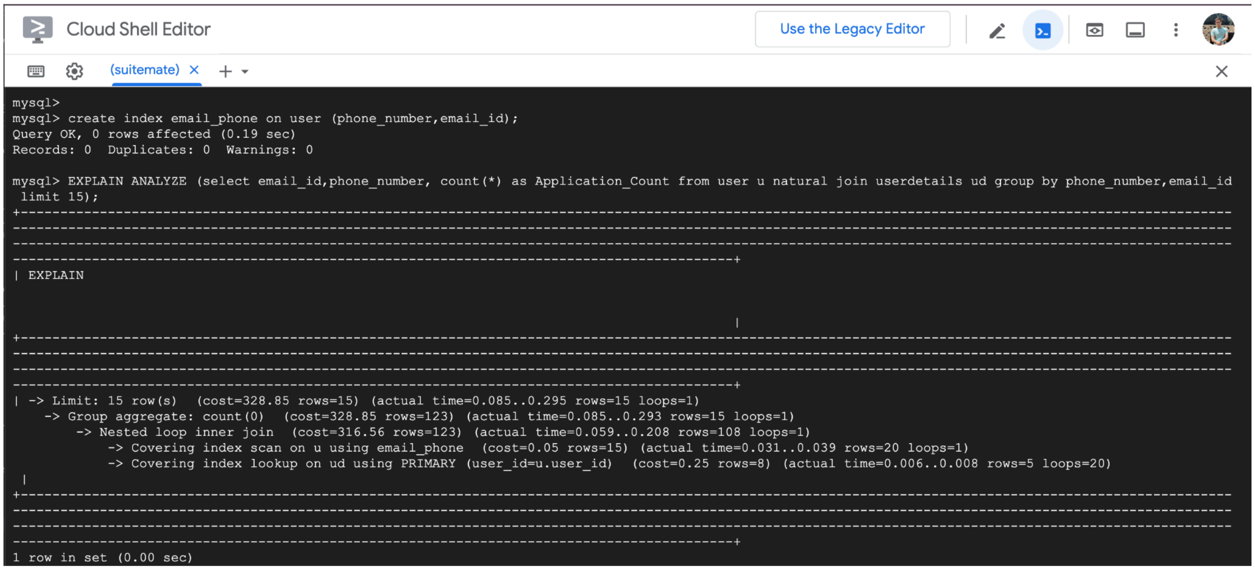

```
mysql>
mysql> select email_id,phone_number, count(*) as Application_Count from user u natural join userdetails ud group by phone_number,email_id limit 15;
+---------------------------+-----------------+-------------------+
| email_id                  | phone_number    | Application_Count |
+---------------------------+-----------------+-------------------+
| clarence45@example.org    | +1 948-717-9195 |                 6 |
| joshua91@example.org      | +1 553-652-5282 |                11 |
| matthew87@example.com     | +1 371-279-3328 |                12 |
| melissa54@example.com     | +1 012-482-9535 |                12 |
| johnsonjennifer@example.net | +1 328-890-7337 |                10 |
| steven72@example.org      | +1 980-513-2659 |                10 |
| xwolf@example.com         | +1 170-987-0286 |                14 |
| perezscott@example.com    | +1 742-183-4504 |                 8 |
| angelica90@example.com    | +1 400-146-4530 |                13 |
| ashleyconrad@example.com  | +1 803-444-7903 |                 9 |
| grahamrobert@example.org  | +1 511-653-3323 |                 9 |
| conradwhitney@example.org | +1 418-501-0979 |                 7 |
| justin05@example.net      | +1 967-389-8536 |                 5 |
| sloanstephanie@example.com | +1 059-495-8531 |                 9 |
| karencurtis@example.org   | +1 196-984-2766 |                14 |
+---------------------------+-----------------+-------------------+
15 rows in set (0.02 sec)
```

Default index:

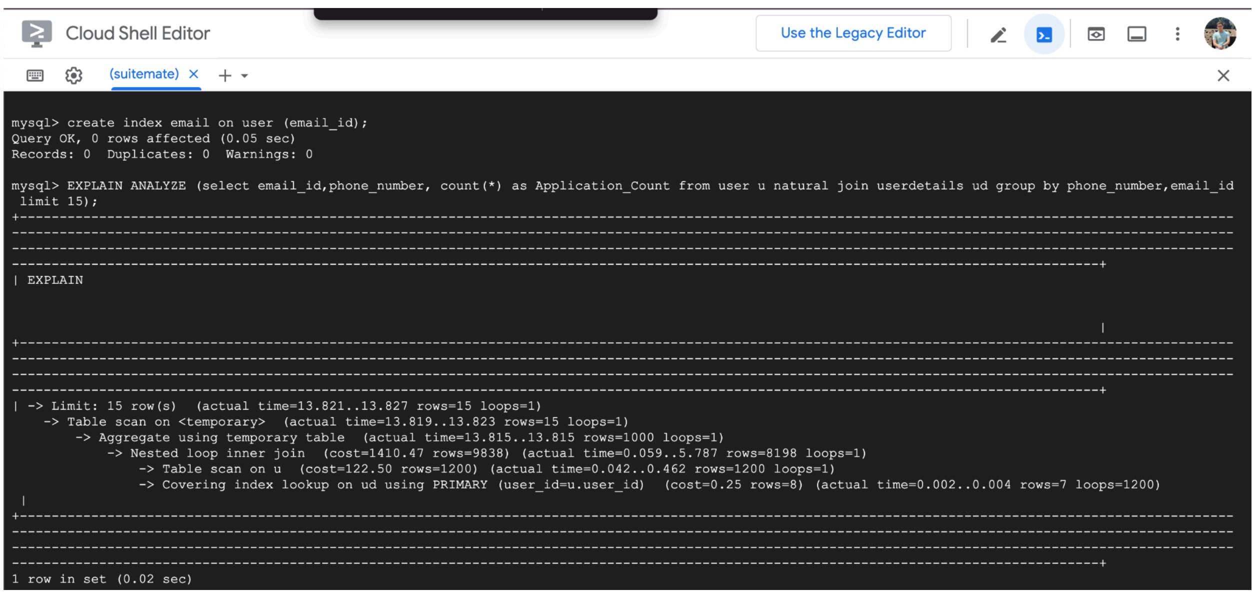
```
mysql> EXPLAIN ANALYZE (select email_id, phone_number, count(*) as Application_Count from user u natural join userdetails ud group by phone_number,email_i
d order by Application_Count desc limit 15);
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                                                                                          |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=14.388..14.390 rows=15 loops=1)
    -> Sort: Application_Count DESC, limit input to 15 row(s) per chunk  (actual time=14.386..14.387 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=13.996..14.235 rows=1000 loops=1)
            -> Aggregate using temporary table  (actual time=13.993..13.993 rows=1000 loops=1)
                -> Nested loop inner join  (cost=1410.47 rows=9838) (actual time=0.092..5.910 rows=8198 loops=1)
                    -> Table scan on u  (cost=122.50 rows=1200) (actual time=0.062..0.500 rows=1200 loops=1)
                    -> Covering index lookup on ud using PRIMARY (user_id=u.user_id)  (cost=0.25 rows=8) (actual time=0.002..0.004 rows=7 loops=1200)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.02 sec)
```

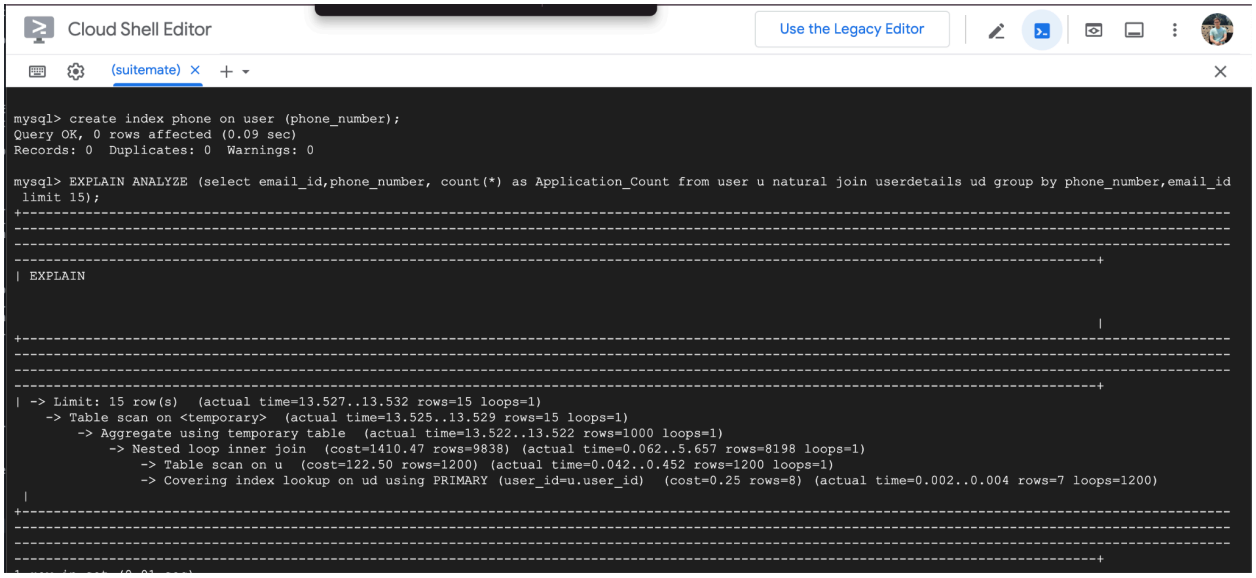
Index on (phone_number, email_id):



Index on email_id:



Index on phone_number:



Analysis

We have tried 3 different combinations of indexing for the above query.

1. Default Index: We first ran the query using the default index and saw that the query engine was doing a table scan using the primary keys as the default index for each table.
2. (phone_number, email_id): After indexing using these two attributes, we could see a significant loss in the cost. Using EXPLAIN ANALYZE, we could see that instead of the table scan, the index range scan reduced the cost significantly.
3. (email_id): Upon indexing with the ‘email_id’ attribute, the cost is the same as compared to the default index. A table scan is being done instead of an index scan.
4. (phone_number): Upon indexing with the ‘phone_number’ attribute, the cost is the same as compared to the default index. A table scan is being done instead of an index scan.

After the above combinations of indexing, indexing using (phone_number, email_id) seems to be improving the overall cost as we are grouping by them. Also, phone_number and email_id together seem to be unique for every user, thus reducing the overall cost.

Complex Query 3:

Query

To determine the minimum, maximum, and average rental prices, along with the corresponding minimum, maximum, and average areas, for properties grouped by pin code. Sort the results by average rent.

```sql
SELECT p.pincode, MIN(u.price) AS Min_Rent, MAX(u.price) AS Max_Rent,
ROUND(AVG(u.price)) AS Avg_Rent, MIN(u.area) as Min_Area, MAX(u.area) AS Max_Area,
ROUND(AVG(u.area)) AS Avg_Area FROM property p NATURAL JOIN unit u GROUP BY
p.pincode LIMIT 15;
```

Cloud Shell Editor | Use the Legacy Editor | (suitemate)

```
mysql> select p.pincode, MIN(u.price) as Min_Rent, MAX(u.price) as Max_Rent, ROUND(AVG(u.price)) as Avg_Rent, MIN(u.area) as Min_Area, MAX(u.area) as Max_
Area, ROUND(AVG(u.area)) as Avg_Area from property p natural join unit u group by p.pincode limit 15;
+---------+----------+----------+----------+----------+----------+----------+
| pincode | Min_Rent | Max_Rent | Avg_Rent | Min_Area | Max_Area | Avg_Area |
+---------+----------+----------+----------+----------+----------+----------+
|   61831 |      800 |     3800 |     1931 |      286 |     2128 |      871 |
|   61854 |      800 |     3800 |     2077 |      277 |     2051 |      947 |
|   61869 |      800 |     3800 |     2242 |      303 |     1814 |     1016 |
|   61838 |      800 |     4000 |     2089 |      314 |     2228 |      986 |
|   61830 |      800 |     3800 |     2170 |      349 |     2062 |     1001 |
|   61857 |      800 |     4000 |     2219 |      308 |     2090 |     1001 |
|   61836 |      800 |     4000 |     2112 |      285 |     2384 |      963 |
|   61862 |      800 |     4000 |     1933 |      265 |     2137 |      892 |
|   61845 |      800 |     4000 |     2367 |      335 |     2015 |     1077 |
|   61842 |      800 |     4000 |     2195 |      301 |     2148 |     1003 |
|   61855 |      800 |     4000 |     2213 |      330 |     2326 |     1015 |
|   61833 |      800 |     4000 |     2005 |      273 |     2225 |      948 |
|   61843 |      800 |     4000 |     2258 |      265 |     1964 |      994 |
|   61834 |      800 |     4000 |     2106 |      275 |     1942 |      972 |
|   61853 |      800 |     3800 |     2212 |      332 |     2044 |     1002 |
+---------+----------+----------+----------+----------+----------+----------+
15 rows in set (0.01 sec)

mysql>
```

Default Index:

Cloud Shell Editor | Use the Legacy Editor | (suitemate)

```
mysql>
mysql>
mysql> EXPLAIN ANALYZE (select p.pincode, MIN(u.price) as Min_Rent, MAX(u.price) as Max_Rent, ROUND(AVG(u.price)) as Avg_Rent, MIN(u.area) as Min_Area, MA
X(u.area) as Max_Area, ROUND(AVG(u.area)) as Avg_Area from property p natural join unit u group by p.pincode limit 15);
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN
                                                                                                                               |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=8.655..8.658 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=8.653..8.655 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=8.650..8.650 rows=51 loops=1)
            -> Nested loop inner join  (cost=1406.07 rows=3667) (actual time=0.108..6.939 rows=3664 loops=1)
                -> Table scan on p  (cost=122.60 rows=1201) (actual time=0.075..0.438 rows=1201 loops=1)
                -> Index lookup on u using property_id (property_id=p.property_id)  (cost=0.76 rows=3) (actual time=0.004..0.005 rows=3 loops=1201)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Index on pincode:

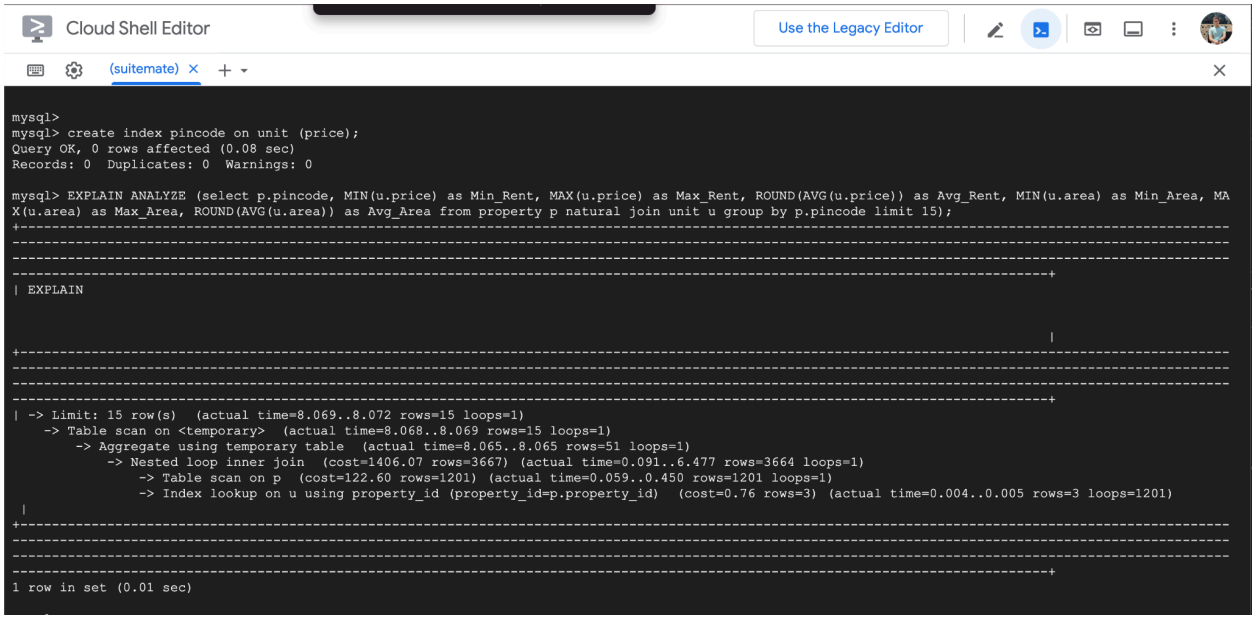

```
    | YES     | NULL       |
| property |          1 | pincode    |           1 | pincode     | A         |          51 |     NULL |   NULL | YES  | BTREE      |         |
    | YES     | NULL       |
+----------+------------+------------+-------------+-------------+-----------+-------------+----------+--------+------+------------+---------+-----------
----+---------+------------+
3 rows in set (0.04 sec)

mysql>
mysql>
mysql> EXPLAIN ANALYZE (select p.pincode, MIN(u.price) as Min_Rent, MAX(u.price) as Max_Rent, ROUND(AVG(u.price)) as Avg_Rent, MIN(u.area) as Min_Area, MA
X(u.area) as Max_Area, ROUND(AVG(u.area)) as Avg_Area from property p natural join unit u group by p.pincode limit 15);
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-------+
| EXPLAIN



       |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-------+
| -> Limit: 15 row(s)  (cost=925.97 rows=15) (actual time=0.242..2.476 rows=15 loops=1)
    -> Group aggregate: avg(u.area), avg(u.price), min(u.price), max(u.price), min(u.area), max(u.area)  (cost=925.97 rows=46) (actual time=0.240..2.473 r
ows=15 loops=1)
        -> Nested loop inner join  (cost=921.39 rows=46) (actual time=0.032..2.093 rows=1079 loops=1)
            -> Covering index scan on p using pincode  (cost=0.05 rows=15) (actual time=0.016..0.106 rows=359 loops=1)
            -> Index lookup on u using property_id (property_id=p.property_id)  (cost=0.76 rows=3) (actual time=0.004..0.005 rows=3 loops=359)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-------+
1 row in set (0.01 sec)
```

Index on price:

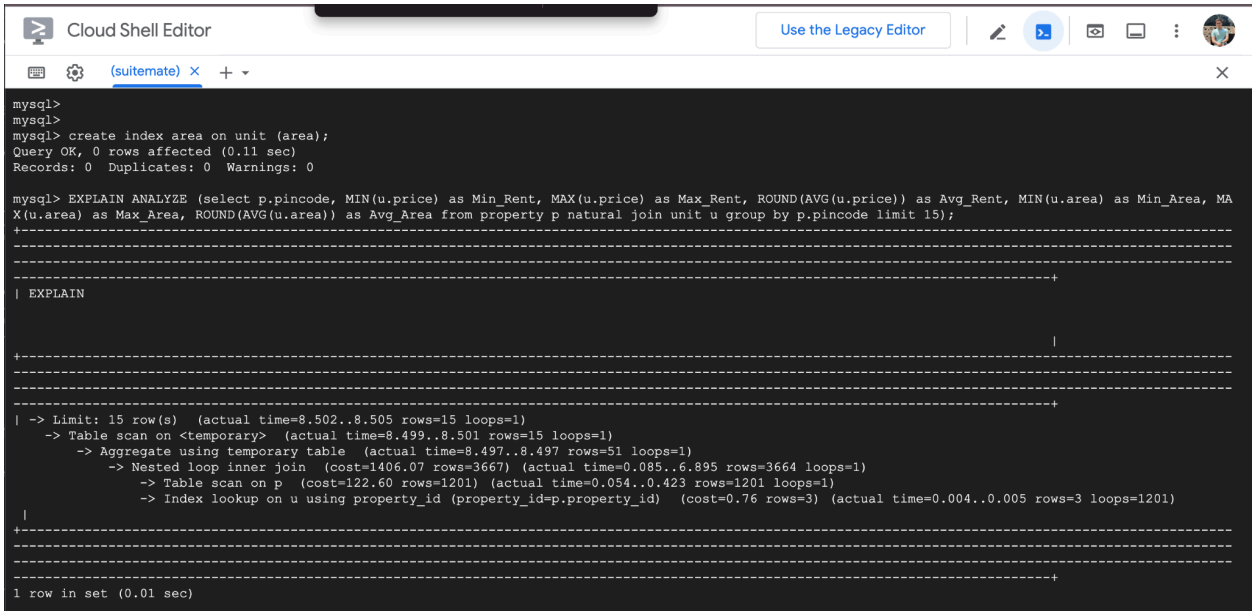



```
mysql>
mysql> create index pincode on unit (price);
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE (select p.pincode, MIN(u.price) as Min_Rent, MAX(u.price) as Max_Rent, ROUND(AVG(u.price)) as Avg_Rent, MIN(u.area) as Min_Area, MA
X(u.area) as Max_Area, ROUND(AVG(u.area)) as Avg_Area from property p natural join unit u group by p.pincode limit 15);
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                                                                                           |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=8.069..8.072 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=8.068..8.069 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=8.065..8.065 rows=51 loops=1)
            -> Nested loop inner join  (cost=1406.07 rows=3667) (actual time=0.091..6.477 rows=3664 loops=1)
                -> Table scan on p  (cost=122.60 rows=1201) (actual time=0.059..0.450 rows=1201 loops=1)
                -> Index lookup on u using property_id (property_id=p.property_id)  (cost=0.76 rows=3) (actual time=0.004..0.005 rows=3 loops=1201)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Index on area:

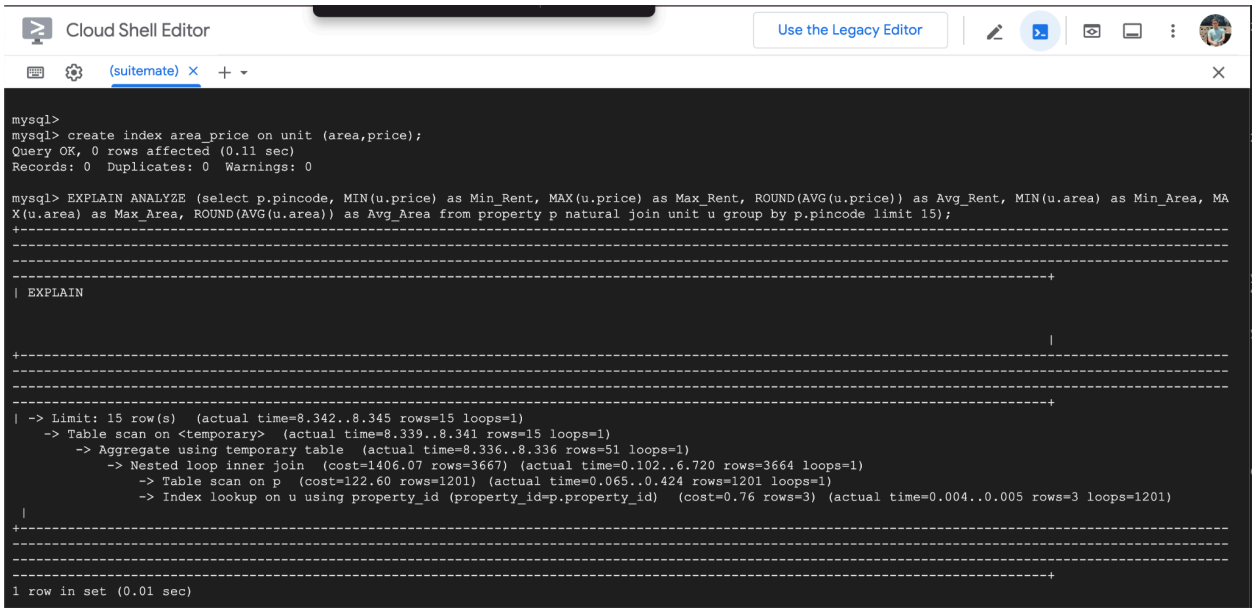



```
mysql>
mysql>
mysql> create index area on unit (area);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE (select p.pincode, MIN(u.price) as Min_Rent, MAX(u.price) as Max_Rent, ROUND(AVG(u.price)) as Avg_Rent, MIN(u.area) as Min_Area, MA
X(u.area) as Max_Area, ROUND(AVG(u.area)) as Avg_Area from property p natural join unit u group by p.pincode limit 15);
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN

                                                                                                                                                          |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=8.502..8.505 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=8.499..8.501 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=8.497..8.497 rows=51 loops=1)
            -> Nested loop inner join  (cost=1406.07 rows=3667) (actual time=0.085..6.895 rows=3664 loops=1)
                -> Table scan on p  (cost=122.60 rows=1201) (actual time=0.054..0.423 rows=1201 loops=1)
                -> Index lookup on u using property_id (property_id=p.property_id)  (cost=0.76 rows=3) (actual time=0.004..0.005 rows=3 loops=1201)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Index on (area, price):




```
mysql>
mysql> create index area_price on unit (area,price);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE (select p.pincode, MIN(u.price) as Min_Rent, MAX(u.price) as Max_Rent, ROUND(AVG(u.price)) as Avg_Rent, MIN(u.area) as Min_Area, MA
X(u.area) as Max_Area, ROUND(AVG(u.area)) as Avg_Area from property p natural join unit u group by p.pincode limit 15);
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN

                                                                                                                                                          |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=8.342..8.345 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=8.339..8.341 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=8.336..8.336 rows=51 loops=1)
            -> Nested loop inner join  (cost=1406.07 rows=3667) (actual time=0.102..6.720 rows=3664 loops=1)
                -> Table scan on p  (cost=122.60 rows=1201) (actual time=0.065..0.424 rows=1201 loops=1)
                -> Index lookup on u using property_id (property_id=p.property_id)  (cost=0.76 rows=3) (actual time=0.004..0.005 rows=3 loops=1201)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Analysis

We have tried 4 different combinations of indexing for the above query.

1. Default Index: We first ran the query using the default index and saw that the query engine was doing a table scan using the primary keys as the default index for each table.
2. (pincode): After indexing using ‘pincode’, we could see a loss in the cost. Using EXPLAIN ANALYZE, we could see that instead of the table scan, the index range scan reduced the cost significantly.
3. (price): Upon indexing with the ‘price’ attribute, the cost was not reduced as compared to the default index, as TableScan is being done.
4. (area): Upon indexing with the ‘area’ attribute, the cost was not reduced as compared to the default index, as TableScan is being done.
5. (area, price): Upon indexing with the ‘area’ and ‘price’ attributes, the cost was not reduced as compared to the default index, as TableScan is being done.

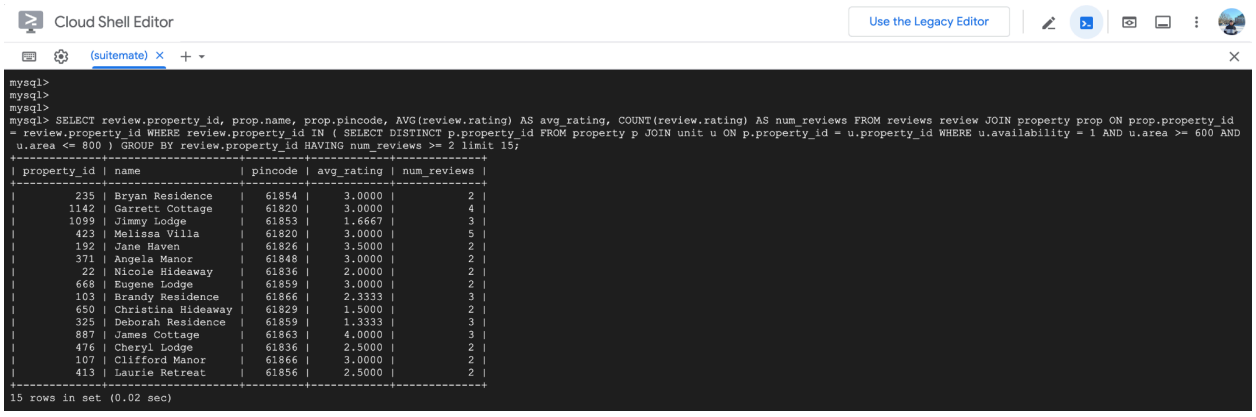
After the above combinations of indexing, indexing using (pincode) seems to be improving the overall cost as we are grouping by them. The 'pincode' has more unique values compared to 'price' or 'area'. When an attribute has high selectivity (i.e., a large number of unique values), indexing can be more beneficial because it helps narrow down the search space efficiently.

Complex Query 4:

Query

To identify the highest-rated properties within specified area requirements by calculating the average ratings and number of reviews. Properties with available units between 600 and 800 square units are considered, with a minimum requirement of two reviews per property.

```
SELECT review.property_id, prop.name, prop.pincode, AVG(review.rating) AS avg_rating, COUNT(review.rating) AS num_reviews FROM reviews review JOIN property prop ON prop.property_id = review.property_id WHERE review.property_id IN ( SELECT DISTINCT p.property_id FROM property p JOIN unit u ON p.property_id = u.property_id WHERE u.availability = 1 AND u.area >= 600 AND u.area <= 800 ) GROUP BY review.property_id HAVING num_reviews >= 2 limit 15;
```



Default Index:

```
| -> Limit: 15 row(s)  (actual time=3.7..3.71 rows=15 loops=1)
    -> Filter: (num_reviews >= 2)  (actual time=3.7..3.71 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=3.7..3.71 rows=33 loops=1)
            -> Aggregate using temporary table  (actual time=3.7..3.7 rows=97 loops=1)
                -> Remove duplicate (prop, Property, review) rows using temporary table (weedout)  (cost=426 rows=77.2) (actual time=0.105..3.51 rows=190 lo
ops=1)
                    -> Nested loop inner join  (cost=426 rows=77.2) (actual time=0.0998..3.41 rows=212 loops=1)
                        -> Nested loop inner join  (cost=399 rows=40.7) (actual time=0.0853..2.5 rows=255 loops=1)
                            -> Nested loop inner join  (cost=385 rows=40.7) (actual time=0.0816..2.13 rows=255 loops=1)
                                -> Filter: ((unit.availability = 1) and (unit.area >= 600) and (unit.area <= 800) and (unit.property_id is not null))  (cost
=370 rows=40.7) (actual time=0.0682..1.65 rows=255 loops=1)
                                    -> Table scan on Unit  (cost=370 rows=3664) (actual time=0.0622..1.31 rows=3664 loops=1)
                                -> Single-row index lookup on prop using PRIMARY (property_id=unit.property_id)  (cost=0.252 rows=1) (actual time=0.00166..0
.00169 rows=1 loops=255)
                            -> Single-row covering index lookup on Property using PRIMARY (property_id=unit.property_id)  (cost=0.252 rows=1) (actual time=0
.0012..0.00124 rows=1 loops=255)
                        -> Index lookup on review using property_id (property_id=unit.property_id)  (cost=0.479 rows=1.9) (actual time=0.00279..0.00332 rows
=0.831 loops=255)
 |
```

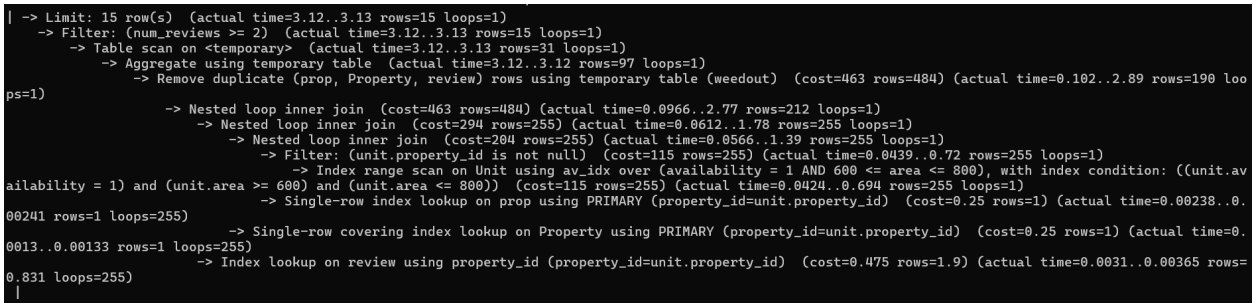
Index on area:

```
| -> Limit: 15 row(s)  (actual time=2.95..2.95 rows=15 loops=1)
    -> Filter: (num_reviews >= 2)  (actual time=2.94..2.95 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=2.94..2.95 rows=31 loops=1)
            -> Aggregate using temporary table  (actual time=2.94..2.94 rows=97 loops=1)
                -> Remove duplicate (prop, Property, review) rows using temporary table (weedout)  (cost=232 rows=74.9) (actual time=0.0857..2.75 rows=190 l
oops=1)
                    -> Nested loop inner join  (cost=232 rows=74.9) (actual time=0.0809..2.64 rows=212 loops=1)
                        -> Nested loop inner join  (cost=206 rows=39.5) (actual time=0.0539..1.72 rows=255 loops=1)
                            -> Nested loop inner join  (cost=192 rows=39.5) (actual time=0.0508..1.33 rows=255 loops=1)
                                -> Filter: ((unit.availability = 1) and (unit.property_id is not null))  (cost=178 rows=39.5) (actual time=0.0405..0.819 row
s=255 loops=1)
                                    -> Index range scan on Unit using area over (600 <= area <= 800), with index condition: ((unit.area >= 600) and (unit.ar
ea <= 800))  (cost=178 rows=395) (actual time=0.0393..0.765 rows=395 loops=1)
                                -> Single-row index lookup on prop using PRIMARY (property_id=unit.property_id)  (cost=0.253 rows=1) (actual time=0.00178..0
.00182 rows=1 loops=255)
                            -> Single-row covering index lookup on Property using PRIMARY (property_id=unit.property_id)  (cost=0.253 rows=1) (actual time=0
.00129..0.00133 rows=1 loops=255)
                        -> Index lookup on review using property_id (property_id=unit.property_id)  (cost=0.479 rows=1.9) (actual time=0.0028..0.00333 rows=
0.831 loops=255)
 |
```

Index on availability:

```
| -> Limit: 15 row(s)  (actual time=5.44..5.45 rows=15 loops=1)
    -> Filter: (num_reviews >= 2)  (actual time=5.44..5.45 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=5.43..5.44 rows=33 loops=1)
            -> Aggregate using temporary table  (actual time=5.43..5.43 rows=97 loops=1)
                -> Remove duplicate (prop, Property, review) rows using temporary table (weedout)  (cost=378 rows=474) (actual time=0.208..5.21 rows=190 loo
ps=1)
                    -> Nested loop inner join  (cost=378 rows=474) (actual time=0.202..5.1 rows=212 loops=1)
                        -> Nested loop inner join  (cost=212 rows=250) (actual time=0.186..4.11 rows=255 loops=1)
                            -> Nested loop inner join  (cost=124 rows=250) (actual time=0.182..3.74 rows=255 loops=1)
                                -> Filter: ((unit.area >= 600) and (unit.area <= 800) and (unit.property_id is not null))  (cost=37 rows=250) (actual time=0
.171..3.2 rows=255 loops=1)
                                    -> Index lookup on Unit using av_idx (availability=1)  (cost=37 rows=2248) (actual time=0.164..3.02 rows=2248 loops=1)
                                -> Single-row index lookup on prop using PRIMARY (property_id=unit.property_id)  (cost=0.25 rows=1) (actual time=0.00188..0.
00191 rows=1 loops=255)
                            -> Single-row covering index lookup on Property using PRIMARY (property_id=unit.property_id)  (cost=0.25 rows=1) (actual time=0.
00122..0.00125 rows=1 loops=255)
                        -> Index lookup on review using property_id (property_id=unit.property_id)  (cost=0.475 rows=1.9) (actual time=0.00308..0.00363 rows
=0.831 loops=255)
 |
```

Index on (availability, area):



Analysis

We have tried 3 different combinations of indexing for the above query.

1. Default Index: We first ran the query using the default index and saw that the query engine was doing a table scan using the primary keys as the default index for each table.
2. (area): After indexing on attribute ‘area’ in the Unit Table, we could see a significant loss in the cost. Using EXPLAIN ANALYZE, we could see that instead of the table scan, the index range scan reduced the cost.
3. (availability): Upon indexing with the ‘availability’ attribute, the cost was reduced as compared to the default index. Using EXPLAIN ANALYZE, we could see that instead of the table scan, index lookup using the availability index reduces the cost significantly.
4. (availability, area): Upon indexing with the combination of ‘availability’ and ‘area’ attributes, the cost reduces slightly, but not as much as when using only a single attribute.

After the above combinations of indexing, indexing using the ‘availability’ attribute seems to be improving the overall cost as using ‘availability’ has a higher impact than the ‘area’ attribute, and filtering based on ‘area’ does not narrow down the result set significantly. Thus ‘availability’ index fetches the rows optimally.