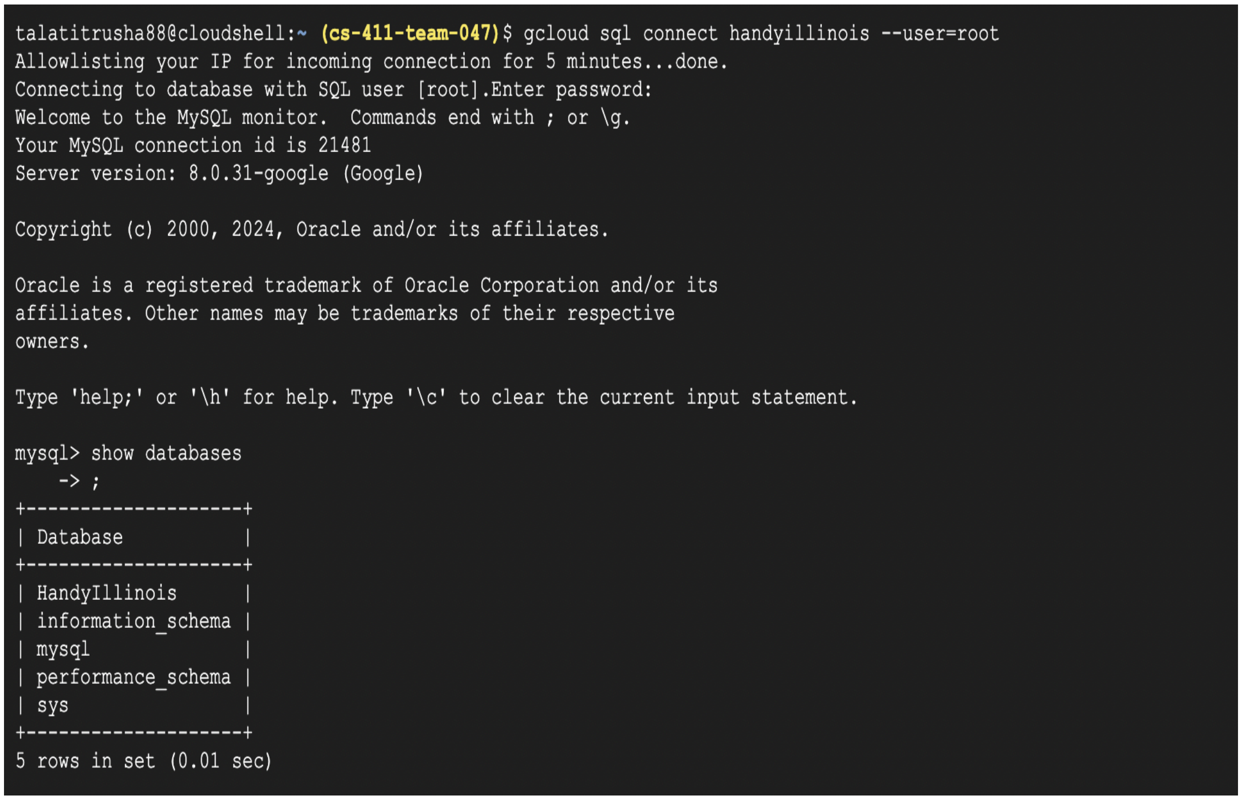
# Project Track 1 Stage 3
# Team 047 - Join_The_Party

## Updated Section: Indexing, Advanced Queries

**<u>Project Title</u>:** HandyIllinois - Connecting Homeowners with Reliable Handyperson Services

## Database implementation :

1. Implementing the database tables on GCP:



```
talatitrusha88@cloudshell:~ (cs-411-team-047)$ gcloud sql connect handyillinois --user=root
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 21481
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases
    -> ;
+--------------------+
| Database           |
+--------------------+
| HandyIllinois      |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)
```

2. DDL Commands for the Tables:

   a. **Agency Table:**

```
CREATE TABLE Agency (
    AgencyId INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255) NOT NULL,
    HQLocation VARCHAR(255),
    Contact VARCHAR(255)
);
```

   b. **Customer Table:**

```
CREATE TABLE Customer (
    CustomerId INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255) NOT NULL,
    Address TEXT,
    ZIP VARCHAR(10),
    ContactNumber VARCHAR(255),
    Password VARCHAR(255),
    Email VARCHAR(255) UNIQUE,
    ProfileImage BLOB
);
```

   c. **Handyperson table:**

```
CREATE TABLE Handyperson (
    HandyId INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255) NOT NULL,
    Skills TEXT,
    Rating DECIMAL(3,1),
    Contact VARCHAR(255),
    ProfileImage BLOB,
    AgencyId INT,
    FOREIGN KEY (AgencyId) REFERENCES Agency(AgencyId) ON
DELETE CASCADE
);
```

d. **Manager table:**

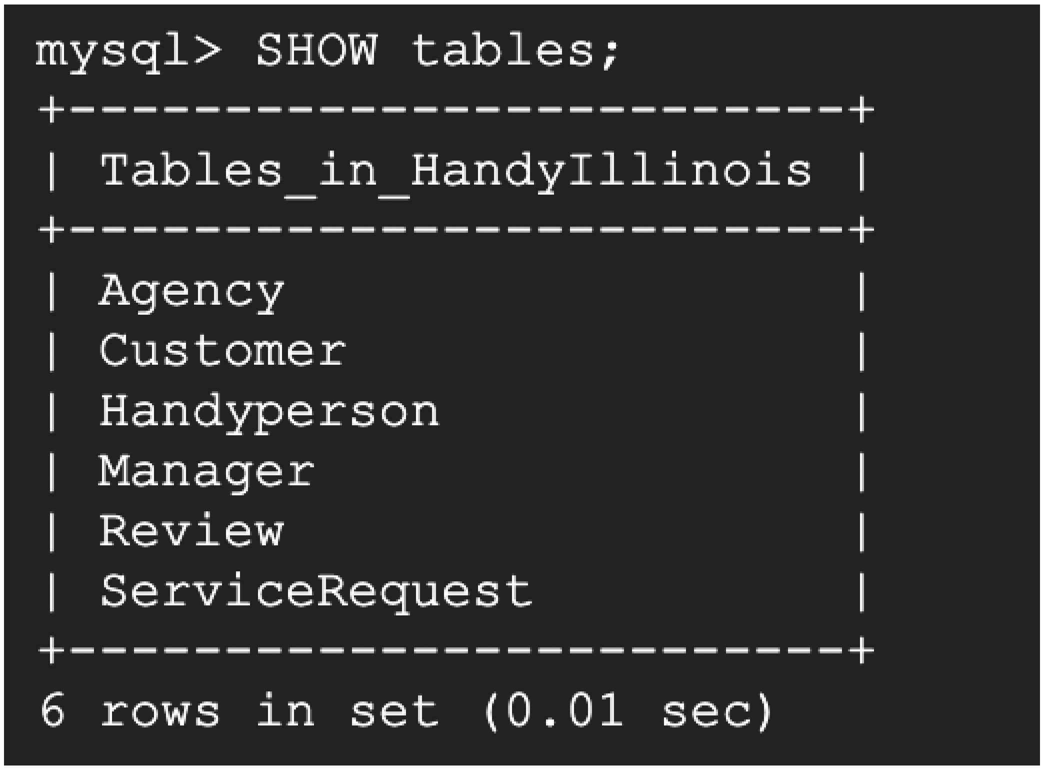
```
CREATE TABLE Manager (
    ManagerId INT PRIMARY KEY AUTO_INCREMENT,
    ManagerName VARCHAR(255) NOT NULL,
    Contact VARCHAR(255),
    ProfileImage BLOB,
    AgencyId INT,
    FOREIGN KEY (AgencyId) REFERENCES Agency(AgencyId) ON
DELETE CASCADE
);
```

e. **Review table:**

```
CREATE TABLE Review (
    ReviewId INT PRIMARY KEY AUTO_INCREMENT,
    Comment TEXT,
    ReviewTitle VARCHAR(255),
    Rating DECIMAL(2,1),
    Date DATE,
    Time TIME,
    CustomerId INT,
    HandyId INT,
    ServiceRequestID INT,
    FOREIGN KEY (CustomerId) REFERENCES Customer(CustomerId)
ON DELETE CASCADE,
    FOREIGN KEY (HandyId) REFERENCES Handyperson(HandyId) ON
DELETE SET NULL,
    FOREIGN KEY (ServiceRequestID) REFERENCES
ServiceRequest(ServiceRequestId) ON DELETE CASCADE
);
```

f. **ServiceRequest Table:**

```
CREATE TABLE ServiceRequest (
    ServiceRequestId INT PRIMARY KEY AUTO_INCREMENT,
    Description TEXT,
    Date DATE,
    Time TIME,
    Status VARCHAR(50),
    Type VARCHAR(255),
    Charges DECIMAL(10,2),
    CustomerId INT,
    ManagerId INT,
    HandyId INT,
    FOREIGN KEY (CustomerId) REFERENCES Customer(CustomerId)
ON DELETE CASCADE,
    FOREIGN KEY (ManagerId) REFERENCES Manager(ManagerId) ON
DELETE SET NULL,
    FOREIGN KEY (HandyId) REFERENCES Handyperson(HandyId) ON
DELETE SET NULL
);
```



```
mysql> SHOW tables;
+-------------------------+
| Tables_in_HandyIllinois |
+-------------------------+
| Agency                  |
| Customer                |
| Handyperson             |
| Manager                 |
| Review                  |
| ServiceRequest          |
+-------------------------+
6 rows in set (0.01 sec)
```

3. Count query:

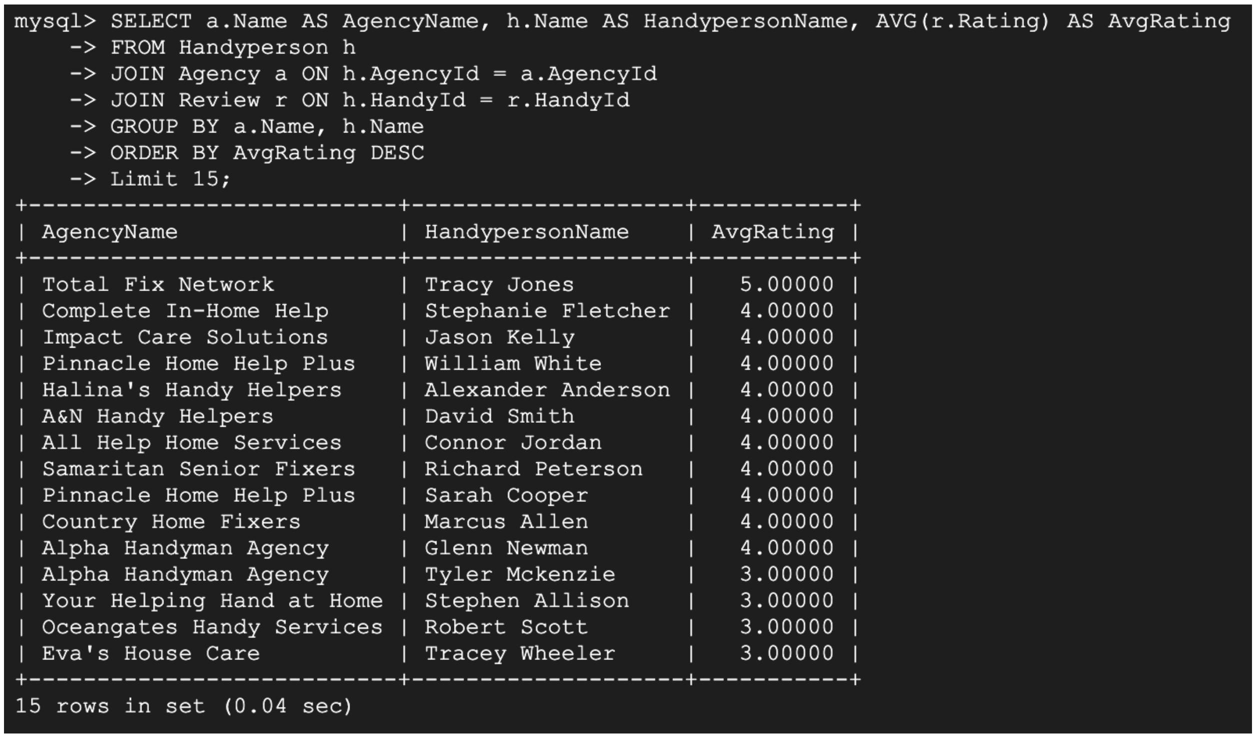
Three tables - Customers, HandyPerson and ServiceRequest have a row count of 1000.

## Advanced Queries:

### Query 1: Retrieve the Average Rating for Each Handyperson by Agency

This query finds the average rating for each Handyperson, grouped by Agency.

```
SELECT a.Name AS AgencyName, h.Name AS HandypersonName, AVG(r.Rating)
AS AvgRating
FROM Handyperson h
JOIN Agency a ON h.AgencyId = a.AgencyId
JOIN Review r ON h.HandyId = r.HandyId
GROUP BY a.Name, h.Name
ORDER BY AvgRating DESC;
```



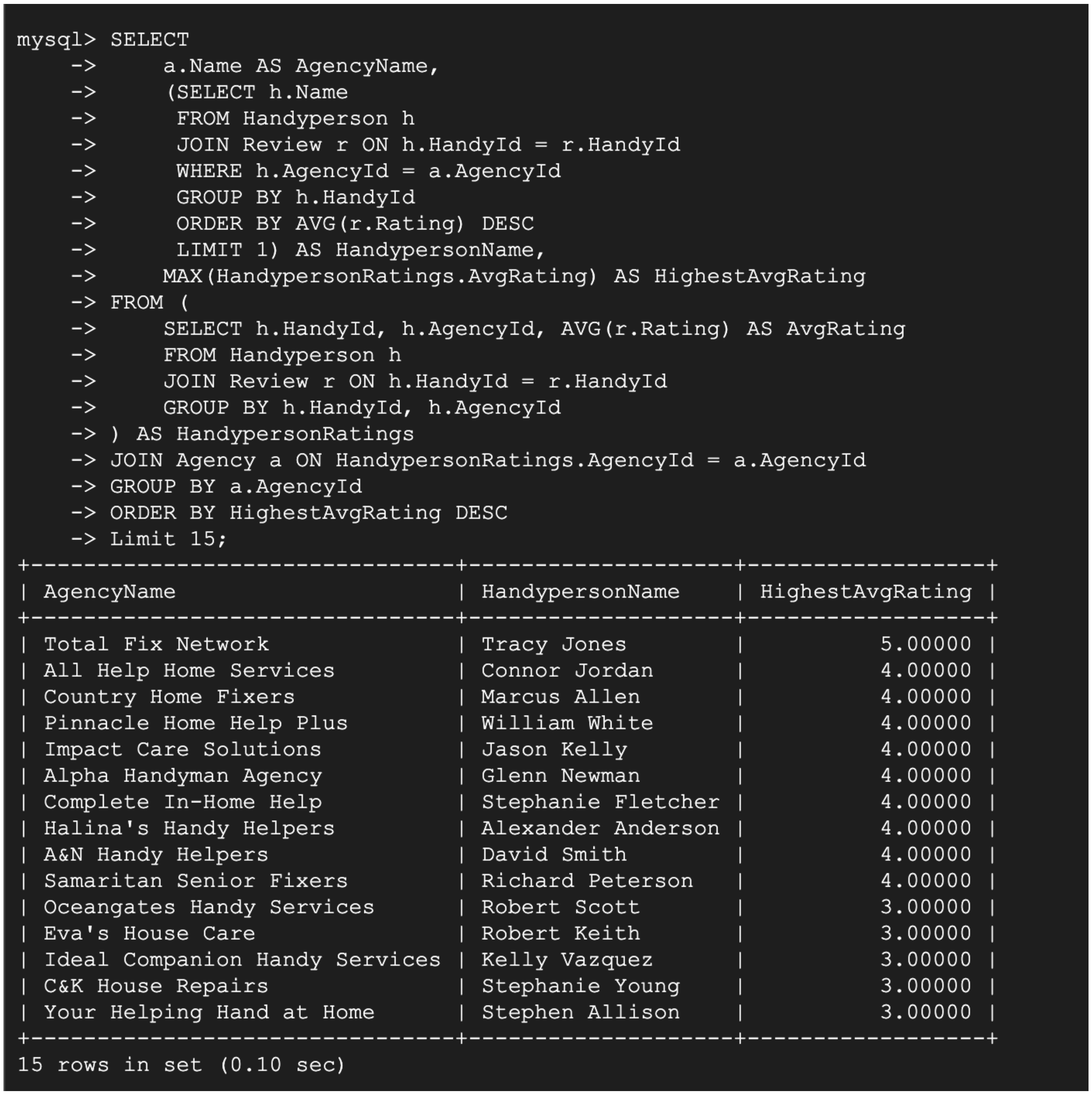
```
mysql> SELECT a.Name AS AgencyName, h.Name AS HandypersonName, AVG(r.Rating) AS AvgRating
    -> FROM Handyperson h
    -> JOIN Agency a ON h.AgencyId = a.AgencyId
    -> JOIN Review r ON h.HandyId = r.HandyId
    -> GROUP BY a.Name, h.Name
    -> ORDER BY AvgRating DESC
    -> Limit 15;
+---------------------------+--------------------+-----------+
| AgencyName                | HandypersonName    | AvgRating |
+---------------------------+--------------------+-----------+
| Total Fix Network         | Tracy Jones        |   5.00000 |
| Complete In-Home Help     | Stephanie Fletcher |   4.00000 |
| Impact Care Solutions     | Jason Kelly        |   4.00000 |
| Pinnacle Home Help Plus   | William White      |   4.00000 |
| Halina's Handy Helpers    | Alexander Anderson |   4.00000 |
| A&N Handy Helpers         | David Smith        |   4.00000 |
| All Help Home Services    | Connor Jordan      |   4.00000 |
| Samaritan Senior Fixers   | Richard Peterson   |   4.00000 |
| Pinnacle Home Help Plus   | Sarah Cooper       |   4.00000 |
| Country Home Fixers       | Marcus Allen       |   4.00000 |
| Alpha Handyman Agency     | Glenn Newman       |   4.00000 |
| Alpha Handyman Agency     | Tyler Mckenzie     |   3.00000 |
| Your Helping Hand at Home | Stephen Allison    |   3.00000 |
| Oceangates Handy Services | Robert Scott       |   3.00000 |
| Eva's House Care          | Tracey Wheeler     |   3.00000 |
+---------------------------+--------------------+-----------+
15 rows in set (0.04 sec)
```

*Result of the first 15 rows*

### Query 2: Retrieve the Highest-Rated Handyperson by Agency

This query finds the handyperson with the highest average rating for each agency, utilizing aggregation and subqueries.

```
SELECT
    a.Name AS AgencyName,
    (SELECT h.Name
     FROM Handyperson h
     JOIN Review r ON h.HandyId = r.HandyId
     WHERE h.AgencyId = a.AgencyId
     GROUP BY h.HandyId
     ORDER BY AVG(r.Rating) DESC
     LIMIT 1) AS HandypersonName,
    MAX(HandypersonRatings.AvgRating) AS HighestAvgRating
FROM (
    SELECT h.HandyId, h.AgencyId, AVG(r.Rating) AS AvgRating
    FROM Handyperson h
    JOIN Review r ON h.HandyId = r.HandyId
    GROUP BY h.HandyId, h.AgencyId
) AS HandypersonRatings
JOIN Agency a ON HandypersonRatings.AgencyId = a.AgencyId
GROUP BY a.AgencyId
ORDER BY HighestAvgRating DESC;
```



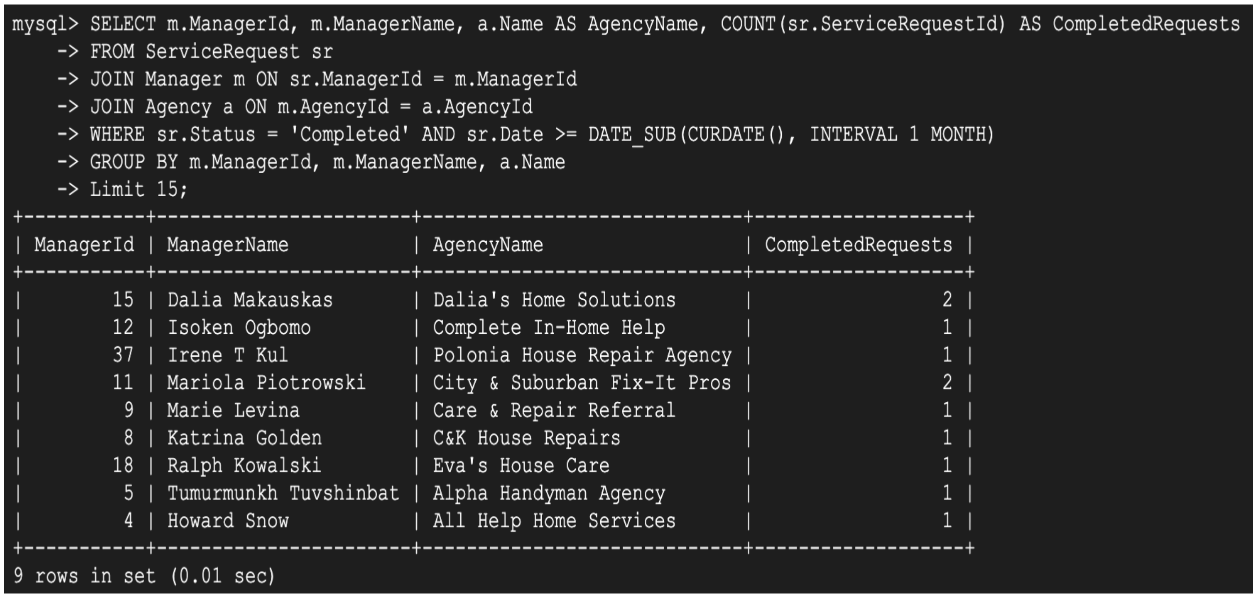
```
mysql> SELECT
    ->     a.Name AS AgencyName,
    ->     (SELECT h.Name
    ->      FROM Handyperson h
    ->      JOIN Review r ON h.HandyId = r.HandyId
    ->      WHERE h.AgencyId = a.AgencyId
    ->      GROUP BY h.HandyId
    ->      ORDER BY AVG(r.Rating) DESC
    ->      LIMIT 1) AS HandypersonName,
    ->     MAX(HandypersonRatings.AvgRating) AS HighestAvgRating
    -> FROM (
    ->     SELECT h.HandyId, h.AgencyId, AVG(r.Rating) AS AvgRating
    ->     FROM Handyperson h
    ->     JOIN Review r ON h.HandyId = r.HandyId
    ->     GROUP BY h.HandyId, h.AgencyId
    -> ) AS HandypersonRatings
    -> JOIN Agency a ON HandypersonRatings.AgencyId = a.AgencyId
    -> GROUP BY a.AgencyId
    -> ORDER BY HighestAvgRating DESC
    -> Limit 15;
+---------------------------------+---------------------+------------------+
| AgencyName                      | HandypersonName     | HighestAvgRating |
+---------------------------------+---------------------+------------------+
| Total Fix Network               | Tracy Jones         |          5.00000 |
| All Help Home Services          | Connor Jordan       |          4.00000 |
| Country Home Fixers             | Marcus Allen        |          4.00000 |
| Pinnacle Home Help Plus         | William White       |          4.00000 |
| Impact Care Solutions           | Jason Kelly         |          4.00000 |
| Alpha Handyman Agency           | Glenn Newman        |          4.00000 |
| Complete In-Home Help           | Stephanie Fletcher  |          4.00000 |
| Halina's Handy Helpers          | Alexander Anderson  |          4.00000 |
| A&N Handy Helpers               | David Smith         |          4.00000 |
| Samaritan Senior Fixers         | Richard Peterson    |          4.00000 |
| Oceangates Handy Services       | Robert Scott        |          3.00000 |
| Eva's House Care                | Robert Keith        |          3.00000 |
| Ideal Companion Handy Services  | Kelly Vazquez       |          3.00000 |
| C&K House Repairs               | Stephanie Young     |          3.00000 |
| Your Helping Hand at Home       | Stephen Allison     |          3.00000 |
+---------------------------------+---------------------+------------------+
15 rows in set (0.10 sec)
```

*Result of the first 15 rows*

### Query 3: Retrieve Service Requests Completed Within the Last Month, Grouped by Manager

This query lists the number of ServiceRequests completed in the past month, grouped by each Manager. It also displays the Manager's associated Agency.

```
SELECT m.ManagerId, m.ManagerName, a.Name AS AgencyName,
COUNT(sr.ServiceRequestId) AS CompletedRequests
FROM ServiceRequest sr
JOIN Manager m ON sr.ManagerId = m.ManagerId
JOIN Agency a ON m.AgencyId = a.AgencyId
WHERE sr.Status = 'Completed' AND sr.Date >= DATE_SUB(CURDATE(),
INTERVAL 1 MONTH)
GROUP BY m.ManagerId, m.ManagerName, a.Name;
```



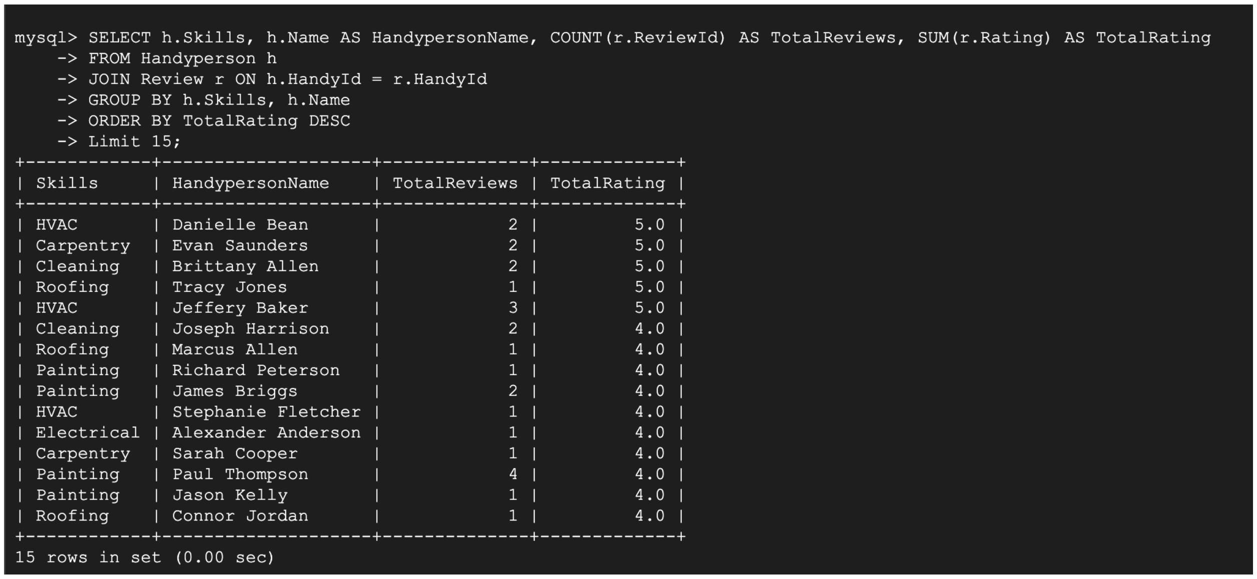
```
mysql> SELECT m.ManagerId, m.ManagerName, a.Name AS AgencyName, COUNT(sr.ServiceRequestId) AS CompletedRequests
    -> FROM ServiceRequest sr
    -> JOIN Manager m ON sr.ManagerId = m.ManagerId
    -> JOIN Agency a ON m.AgencyId = a.AgencyId
    -> WHERE sr.Status = 'Completed' AND sr.Date >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
    -> GROUP BY m.ManagerId, m.ManagerName, a.Name
    -> Limit 15;
+-----------+----------------------+-----------------------------+-------------------+
| ManagerId | ManagerName          | AgencyName                  | CompletedRequests |
+-----------+----------------------+-----------------------------+-------------------+
|        15 | Dalia Makauskas      | Dalia's Home Solutions      |                 2 |
|        12 | Isoken Ogbomo        | Complete In-Home Help       |                 1 |
|        37 | Irene T Kul          | Polonia House Repair Agency |                 1 |
|        11 | Mariola Piotrowski   | City & Suburban Fix-It Pros |                 2 |
|         9 | Marie Levina         | Care & Repair Referral      |                 1 |
|         8 | Katrina Golden       | C&K House Repairs           |                 1 |
|        18 | Ralph Kowalski       | Eva's House Care            |                 1 |
|         5 | Tumurmunkh Tuvshinbat | Alpha Handyman Agency      |                 1 |
|         4 | Howard Snow          | All Help Home Services      |                 1 |
+-----------+----------------------+-----------------------------+-------------------+
9 rows in set (0.01 sec)
```

***Result of the first 9 rows* (Due to number of managers being limited)**

### Query 4: Retrieve Total Ratings and Number of Reviews for Each Handyperson, Grouped by Skill

This query groups handypersons by skill and calculates their total ratings and the number of reviews they've received, involving aggregation and a join.

```
SELECT h.Skills, h.Name AS HandypersonName, COUNT(r.ReviewId) AS
TotalReviews, SUM(r.Rating) AS TotalRating
FROM Handyperson h
JOIN Review r ON h.HandyId = r.HandyId
GROUP BY h.Skills, h.Name
ORDER BY TotalRating DESC;
```



```
mysql> SELECT h.Skills, h.Name AS HandypersonName, COUNT(r.ReviewId) AS TotalReviews, SUM(r.Rating) AS TotalRating
    -> FROM Handyperson h
    -> JOIN Review r ON h.HandyId = r.HandyId
    -> GROUP BY h.Skills, h.Name
    -> ORDER BY TotalRating DESC
    -> Limit 15;
+------------+--------------------+--------------+-------------+
| Skills     | HandypersonName    | TotalReviews | TotalRating |
+------------+--------------------+--------------+-------------+
| HVAC       | Danielle Bean      |            2 |         5.0 |
| Carpentry  | Evan Saunders      |            2 |         5.0 |
| Cleaning   | Brittany Allen     |            2 |         5.0 |
| Roofing    | Tracy Jones        |            1 |         5.0 |
| HVAC       | Jeffery Baker      |            3 |         5.0 |
| Cleaning   | Joseph Harrison    |            2 |         4.0 |
| Roofing    | Marcus Allen       |            1 |         4.0 |
| Painting   | Richard Peterson   |            1 |         4.0 |
| Painting   | James Briggs       |            2 |         4.0 |
| HVAC       | Stephanie Fletcher |            1 |         4.0 |
| Electrical | Alexander Anderson |            1 |         4.0 |
| Carpentry  | Sarah Cooper       |            1 |         4.0 |
| Painting   | Paul Thompson      |            4 |         4.0 |
| Painting   | Jason Kelly        |            1 |         4.0 |
| Roofing    | Connor Jordan      |            1 |         4.0 |
+------------+--------------------+--------------+-------------+
15 rows in set (0.00 sec)
```

*Result of the first 15 rows*

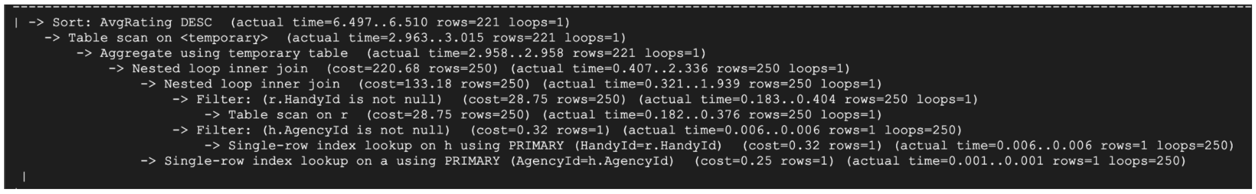
## Indexing:

### Query 1: Retrieve the Average Rating for Each Handyperson by Agency

This query finds the average rating for each Handyperson, grouped by Agency.

```
SELECT a.Name AS AgencyName, h.Name AS HandypersonName, AVG(r.Rating)
AS AvgRating
FROM Handyperson h
JOIN Agency a ON h.AgencyId = a.AgencyId
JOIN Review r ON h.HandyId = r.HandyId
GROUP BY a.Name, h.Name
ORDER BY AvgRating DESC;
```

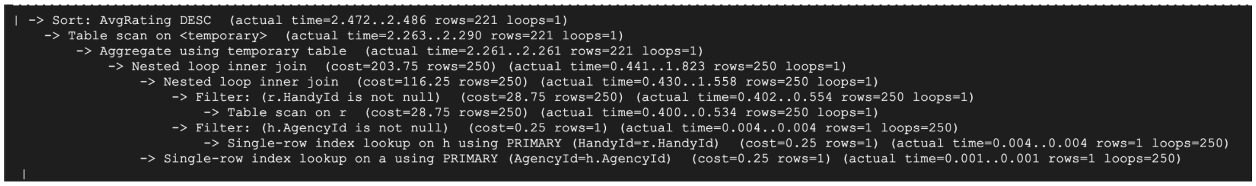
**Before Indexing:**



```
| -> Sort: AvgRating DESC  (actual time=6.497..6.510 rows=221 loops=1)
    -> Table scan on <temporary>  (actual time=2.963..3.015 rows=221 loops=1)
        -> Aggregate using temporary table  (actual time=2.958..2.958 rows=221 loops=1)
            -> Nested loop inner join  (cost=220.68 rows=250) (actual time=0.407..2.336 rows=250 loops=1)
                -> Nested loop inner join  (cost=133.18 rows=250) (actual time=0.321..1.939 rows=250 loops=1)
                    -> Filter: (r.HandyId is not null)  (cost=28.75 rows=250) (actual time=0.183..0.404 rows=250 loops=1)
                        -> Table scan on r  (cost=28.75 rows=250) (actual time=0.182..0.376 rows=250 loops=1)
                    -> Filter: (h.AgencyId is not null)  (cost=0.32 rows=1) (actual time=0.006..0.006 rows=1 loops=250)
                        -> Single-row index lookup on h using PRIMARY (HandyId=r.HandyId)  (cost=0.32 rows=1) (actual time=0.006..0.006 rows=1 loops=250)
                -> Single-row index lookup on a using PRIMARY (AgencyId=h.AgencyId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=250)
 |
```

**After Indexing:**

1. **Create index handy_rating on Handyperson(Rating)**



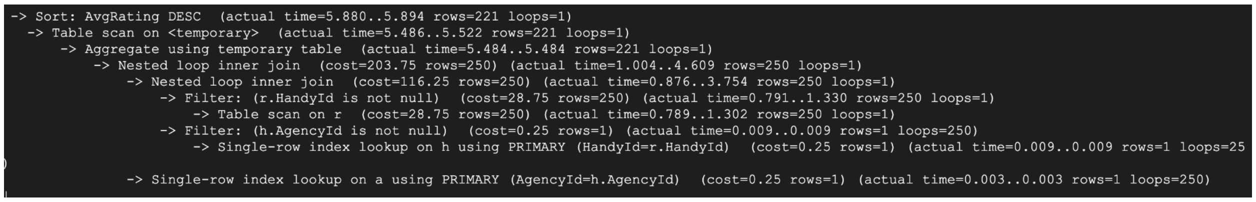
```
| -> Sort: AvgRating DESC  (actual time=2.472..2.486 rows=221 loops=1)
    -> Table scan on <temporary>  (actual time=2.263..2.290 rows=221 loops=1)
        -> Aggregate using temporary table  (actual time=2.261..2.261 rows=221 loops=1)
            -> Nested loop inner join  (cost=203.75 rows=250) (actual time=0.441..1.823 rows=250 loops=1)
                -> Nested loop inner join  (cost=116.25 rows=250) (actual time=0.430..1.558 rows=250 loops=1)
                    -> Filter: (r.HandyId is not null)  (cost=28.75 rows=250) (actual time=0.402..0.554 rows=250 loops=1)
                        -> Table scan on r  (cost=28.75 rows=250) (actual time=0.400..0.534 rows=250 loops=1)
                    -> Filter: (h.AgencyId is not null)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=250)
                        -> Single-row index lookup on h using PRIMARY (HandyId=r.HandyId)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=250)
                -> Single-row index lookup on a using PRIMARY (AgencyId=h.AgencyId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=250)
 |
```

Cost for both inner joins reduced from

**220.68** -> **203.75**

**133** -> **116.25**

2. **Create index handy_name on Handyperson(Name);**

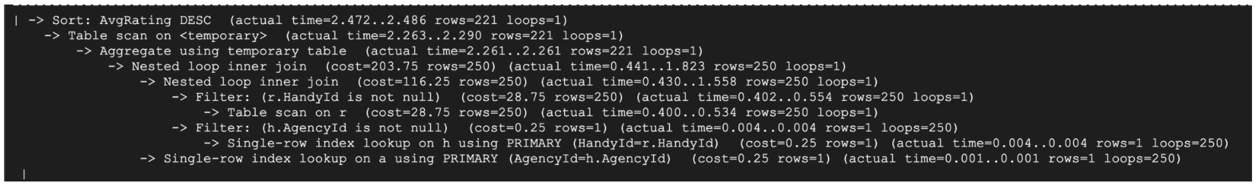


```
-> Sort: AvgRating DESC  (actual time=5.880..5.894 rows=221 loops=1)
  -> Table scan on <temporary>  (actual time=5.486..5.522 rows=221 loops=1)
      -> Aggregate using temporary table  (actual time=5.484..5.484 rows=221 loops=1)
          -> Nested loop inner join  (cost=203.75 rows=250) (actual time=1.004..4.609 rows=250 loops=1)
              -> Nested loop inner join  (cost=116.25 rows=250) (actual time=0.876..3.754 rows=250 loops=1)
                  -> Filter: (r.HandyId is not null)  (cost=28.75 rows=250) (actual time=0.791..1.330 rows=250 loops=1)
                      -> Table scan on r  (cost=28.75 rows=250) (actual time=0.789..1.302 rows=250 loops=1)
                  -> Filter: (h.AgencyId is not null)  (cost=0.25 rows=1) (actual time=0.009..0.009 rows=1 loops=250)
                      -> Single-row index lookup on h using PRIMARY (HandyId=r.HandyId)  (cost=0.25 rows=1) (actual time=0.009..0.009 rows=1 loops=25
)

              -> Single-row index lookup on a using PRIMARY (AgencyId=h.AgencyId)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=250)
```

No change in cost

3. **create index agency_name on Agency(Name);**



```
| -> Sort: AvgRating DESC  (actual time=2.472..2.486 rows=221 loops=1)
    -> Table scan on <temporary>  (actual time=2.263..2.290 rows=221 loops=1)
        -> Aggregate using temporary table  (actual time=2.261..2.261 rows=221 loops=1)
            -> Nested loop inner join  (cost=203.75 rows=250) (actual time=0.441..1.823 rows=250 loops=1)
                -> Nested loop inner join  (cost=116.25 rows=250) (actual time=0.430..1.558 rows=250 loops=1)
                    -> Filter: (r.HandyId is not null)  (cost=28.75 rows=250) (actual time=0.402..0.554 rows=250 loops=1)
                        -> Table scan on r  (cost=28.75 rows=250) (actual time=0.400..0.534 rows=250 loops=1)
                    -> Filter: (h.AgencyId is not null)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=250)
                        -> Single-row index lookup on h using PRIMARY (HandyId=r.HandyId)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=250)
                -> Single-row index lookup on a using PRIMARY (AgencyId=h.AgencyId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=250)
 |
```

No change in cost

**Final index design:**

1. **`Create index handy_rating on Handyperson(Rating)`**

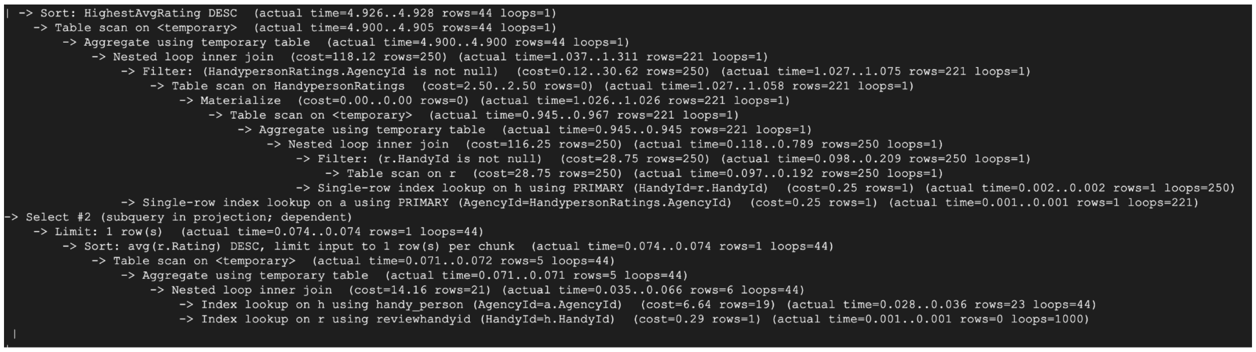
Creating an index on the Rating attribute on handyperson table has led to better inner join costs. Other than the fact that all the other attributes that are involved in the joins are pre indexed by default due to being primary and foreign keys, Rating attribute which is used to aggregate the average is the only attribute which could be indexed and would give a efficient cost.

## Query 2: Retrieve the Highest-Rated Handyperson by Agency

This query finds the handyperson with the highest average rating for each agency, utilizing aggregation and subqueries.

```
SELECT
    a.Name AS AgencyName,
    (SELECT h.Name
     FROM Handyperson h
     JOIN Review r ON h.HandyId = r.HandyId
     WHERE h.AgencyId = a.AgencyId
     GROUP BY h.HandyId
     ORDER BY AVG(r.Rating) DESC
     LIMIT 1) AS HandypersonName,
    MAX(HandypersonRatings.AvgRating) AS HighestAvgRating
FROM (
    SELECT h.HandyId, h.AgencyId, AVG(r.Rating) AS AvgRating
    FROM Handyperson h
    JOIN Review r ON h.HandyId = r.HandyId
    GROUP BY h.HandyId, h.AgencyId
) AS HandypersonRatings
JOIN Agency a ON HandypersonRatings.AgencyId = a.AgencyId
GROUP BY a.AgencyId
ORDER BY HighestAvgRating DESC;
```

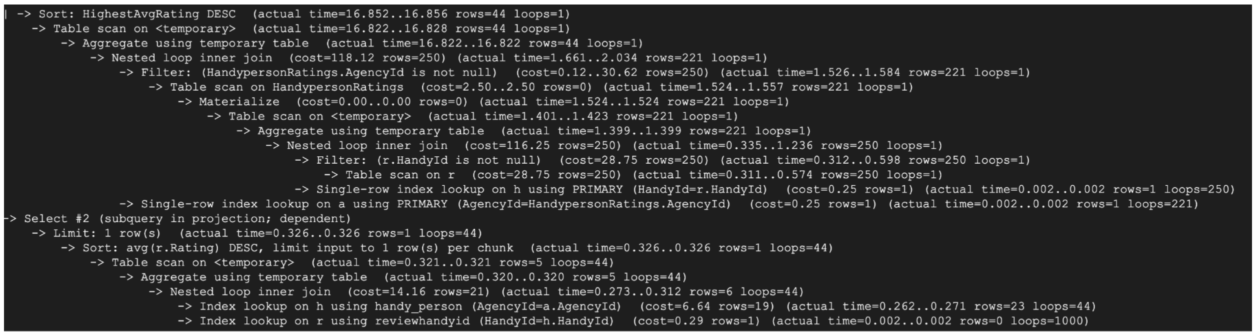
**Before Indexing:**



```
| -> Sort: HighestAvgRating DESC  (actual time=4.926..4.928 rows=44 loops=1)
    -> Table scan on <temporary>  (actual time=4.900..4.905 rows=44 loops=1)
        -> Aggregate using temporary table  (actual time=4.900..4.900 rows=44 loops=1)
            -> Nested loop inner join  (cost=118.12 rows=250) (actual time=1.037..1.311 rows=221 loops=1)
                -> Filter: (HandypersonRatings.AgencyId is not null)  (cost=0.12..30.62 rows=250) (actual time=1.027..1.075 rows=221 loops=1)
                    -> Table scan on HandypersonRatings  (cost=2.50..2.50 rows=0) (actual time=1.027..1.058 rows=221 loops=1)
                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1.026..1.026 rows=221 loops=1)
                            -> Table scan on <temporary>  (actual time=0.945..0.967 rows=221 loops=1)
                                -> Aggregate using temporary table  (actual time=0.945..0.945 rows=221 loops=1)
                                    -> Nested loop inner join  (cost=116.25 rows=250) (actual time=0.118..0.789 rows=250 loops=1)
                                        -> Filter: (r.HandyId is not null)  (cost=28.75 rows=250) (actual time=0.098..0.209 rows=250 loops=1)
                                            -> Table scan on r  (cost=28.75 rows=250) (actual time=0.097..0.192 rows=250 loops=1)
                                        -> Single-row index lookup on h using PRIMARY (HandyId=r.HandyId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=250)
                -> Single-row index lookup on a using PRIMARY (AgencyId=HandypersonRatings.AgencyId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=221)
-> Select #2 (subquery in projection; dependent)
    -> Limit: 1 row(s)  (actual time=0.074..0.074 rows=1 loops=44)
        -> Sort: avg(r.Rating) DESC, limit input to 1 row(s) per chunk  (actual time=0.074..0.074 rows=1 loops=44)
            -> Table scan on <temporary>  (actual time=0.071..0.072 rows=5 loops=44)
                -> Aggregate using temporary table  (actual time=0.071..0.071 rows=5 loops=44)
                    -> Nested loop inner join  (cost=14.16 rows=21) (actual time=0.035..0.066 rows=6 loops=44)
                        -> Index lookup on h using handy_person (AgencyId=a.AgencyId)  (cost=6.64 rows=19) (actual time=0.028..0.036 rows=23 loops=44)
                        -> Index lookup on r using reviewhandyid (HandyId=h.HandyId)  (cost=0.29 rows=1) (actual time=0.001..0.001 rows=0 loops=1000)
 |
```

**After Indexing:**

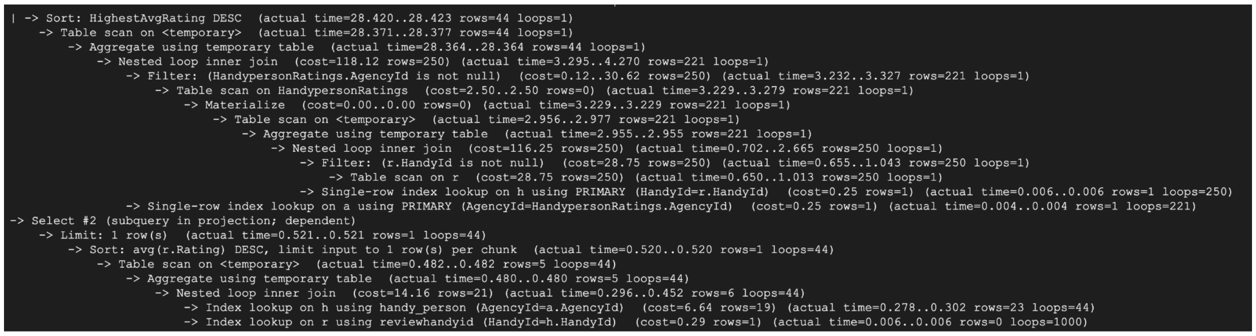
1. **Create index handy_rating on Handyperson(Rating);**



```
| -> Sort: HighestAvgRating DESC  (actual time=16.852..16.856 rows=44 loops=1)
    -> Table scan on <temporary>  (actual time=16.822..16.828 rows=44 loops=1)
        -> Aggregate using temporary table  (actual time=16.822..16.822 rows=44 loops=1)
            -> Nested loop inner join  (cost=118.12 rows=250) (actual time=1.661..2.034 rows=221 loops=1)
                -> Filter: (HandypersonRatings.AgencyId is not null)  (cost=0.12..30.62 rows=250) (actual time=1.526..1.584 rows=221 loops=1)
                    -> Table scan on HandypersonRatings  (cost=2.50..2.50 rows=0) (actual time=1.524..1.557 rows=221 loops=1)
                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1.524..1.524 rows=221 loops=1)
                            -> Table scan on <temporary>  (actual time=1.401..1.423 rows=221 loops=1)
                                -> Aggregate using temporary table  (actual time=1.399..1.399 rows=221 loops=1)
                                    -> Nested loop inner join  (cost=116.25 rows=250) (actual time=0.335..1.236 rows=250 loops=1)
                                        -> Filter: (r.HandyId is not null)  (cost=28.75 rows=250) (actual time=0.312..0.598 rows=250 loops=1)
                                            -> Table scan on r  (cost=28.75 rows=250) (actual time=0.311..0.574 rows=250 loops=1)
                                        -> Single-row index lookup on h using PRIMARY (HandyId=r.HandyId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=250)
                -> Single-row index lookup on a using PRIMARY (AgencyId=HandypersonRatings.AgencyId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=221)
-> Select #2 (subquery in projection; dependent)
    -> Limit: 1 row(s)  (actual time=0.326..0.326 rows=1 loops=44)
        -> Sort: avg(r.Rating) DESC, limit input to 1 row(s) per chunk  (actual time=0.326..0.326 rows=1 loops=44)
            -> Table scan on <temporary>  (actual time=0.321..0.321 rows=5 loops=44)
                -> Aggregate using temporary table  (actual time=0.320..0.320 rows=5 loops=44)
                    -> Nested loop inner join  (cost=14.16 rows=21) (actual time=0.273..0.312 rows=6 loops=44)
                        -> Index lookup on h using handy_person (AgencyId=a.AgencyId)  (cost=6.64 rows=19) (actual time=0.262..0.271 rows=23 loops=44)
                        -> Index lookup on r using reviewhandyid (HandyId=h.HandyId)  (cost=0.29 rows=1) (actual time=0.002..0.002 rows=0 loops=1000)
```

No Change in Cost

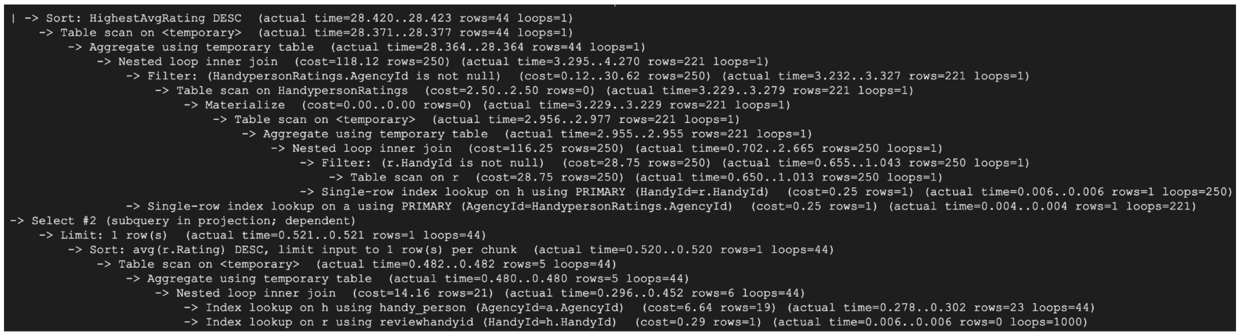
2. **Create index handy_name on Handyperson(Name);**



```
| -> Sort: HighestAvgRating DESC  (actual time=28.420..28.423 rows=44 loops=1)
    -> Table scan on <temporary>  (actual time=28.371..28.377 rows=44 loops=1)
        -> Aggregate using temporary table  (actual time=28.364..28.364 rows=44 loops=1)
            -> Nested loop inner join  (cost=118.12 rows=250) (actual time=3.295..4.270 rows=221 loops=1)
                -> Filter: (HandypersonRatings.AgencyId is not null)  (cost=0.12..30.62 rows=250) (actual time=3.232..3.327 rows=221 loops=1)
                    -> Table scan on HandypersonRatings  (cost=2.50..2.50 rows=0) (actual time=3.229..3.279 rows=221 loops=1)
                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.229..3.229 rows=221 loops=1)
                            -> Table scan on <temporary>  (actual time=2.956..2.977 rows=221 loops=1)
                                -> Aggregate using temporary table  (actual time=2.955..2.955 rows=221 loops=1)
                                    -> Nested loop inner join  (cost=116.25 rows=250) (actual time=0.702..2.665 rows=250 loops=1)
                                        -> Filter: (r.HandyId is not null)  (cost=28.75 rows=250) (actual time=0.655..1.043 rows=250 loops=1)
                                            -> Table scan on r  (cost=28.75 rows=250) (actual time=0.650..1.013 rows=250 loops=1)
                                        -> Single-row index lookup on h using PRIMARY (HandyId=r.HandyId)  (cost=0.25 rows=1) (actual time=0.006..0.006 rows=1 loops=250)
                -> Single-row index lookup on a using PRIMARY (AgencyId=HandypersonRatings.AgencyId)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=221)
-> Select #2 (subquery in projection; dependent)
    -> Limit: 1 row(s)  (actual time=0.521..0.521 rows=1 loops=44)
        -> Sort: avg(r.Rating) DESC, limit input to 1 row(s) per chunk  (actual time=0.520..0.520 rows=1 loops=44)
            -> Table scan on <temporary>  (actual time=0.482..0.482 rows=5 loops=44)
                -> Aggregate using temporary table  (actual time=0.480..0.480 rows=5 loops=44)
                    -> Nested loop inner join  (cost=14.16 rows=21) (actual time=0.296..0.452 rows=6 loops=44)
                        -> Index lookup on h using handy_person (AgencyId=a.AgencyId)  (cost=6.64 rows=19) (actual time=0.278..0.302 rows=23 loops=44)
                        -> Index lookup on r using reviewhandyid (HandyId=h.HandyId)  (cost=0.29 rows=1) (actual time=0.006..0.006 rows=0 loops=1000)
```

No change in Cost

3. **create index agency_name on Agency(Name);**



```
| -> Sort: HighestAvgRating DESC  (actual time=28.420..28.423 rows=44 loops=1)
    -> Table scan on <temporary>  (actual time=28.371..28.377 rows=44 loops=1)
        -> Aggregate using temporary table  (actual time=28.364..28.364 rows=44 loops=1)
            -> Nested loop inner join  (cost=118.12 rows=250) (actual time=3.295..4.270 rows=221 loops=1)
                -> Filter: (HandypersonRatings.AgencyId is not null)  (cost=0.12..30.62 rows=250) (actual time=3.232..3.327 rows=221 loops=1)
                    -> Table scan on HandypersonRatings  (cost=2.50..2.50 rows=0) (actual time=3.229..3.279 rows=221 loops=1)
                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.229..3.229 rows=221 loops=1)
                            -> Table scan on <temporary>  (actual time=2.956..2.977 rows=221 loops=1)
                                -> Aggregate using temporary table  (actual time=2.955..2.955 rows=221 loops=1)
                                    -> Nested loop inner join  (cost=116.25 rows=250) (actual time=0.702..2.665 rows=250 loops=1)
                                        -> Filter: (r.HandyId is not null)  (cost=28.75 rows=250) (actual time=0.655..1.043 rows=250 loops=1)
                                            -> Table scan on r  (cost=28.75 rows=250) (actual time=0.650..1.013 rows=250 loops=1)
                                        -> Single-row index lookup on h using PRIMARY (HandyId=r.HandyId)  (cost=0.25 rows=1) (actual time=0.006..0.006 rows=1 loops=250)
                -> Single-row index lookup on a using PRIMARY (AgencyId=HandypersonRatings.AgencyId)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=221)
-> Select #2 (subquery in projection; dependent)
    -> Limit: 1 row(s)  (actual time=0.521..0.521 rows=1 loops=44)
        -> Sort: avg(r.Rating) DESC, limit input to 1 row(s) per chunk  (actual time=0.520..0.520 rows=1 loops=44)
            -> Table scan on <temporary>  (actual time=0.482..0.482 rows=5 loops=44)
                -> Aggregate using temporary table  (actual time=0.480..0.480 rows=5 loops=44)
                    -> Nested loop inner join  (cost=14.16 rows=21) (actual time=0.296..0.452 rows=6 loops=44)
                        -> Index lookup on h using handy_person (AgencyId=a.AgencyId)  (cost=6.64 rows=19) (actual time=0.278..0.302 rows=23 loops=44)
                        -> Index lookup on r using reviewhandyid (HandyId=h.HandyId)  (cost=0.29 rows=1) (actual time=0.006..0.006 rows=0 loops=1000)
```

No change in cost

**Final index design:**

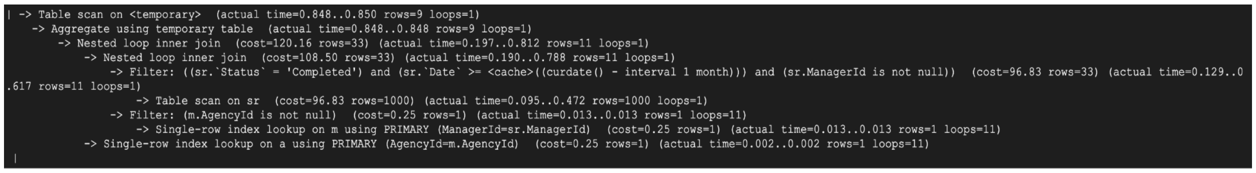
None of the indexes helped due to low cardinality and overhead created by the index itself. In this case even though the joins were complex, indexes didn't help. But rather than bTree indexes,which were applied here, hash table or bitmap indexes may help more in this case.

## Query 3: Retrieve Service Requests Completed Within the Last Month, Grouped by Manager

This query lists the number of ServiceRequests completed in the past month, grouped by each Manager. It also displays the Manager's associated Agency.

```
SELECT m.ManagerId, m.ManagerName, a.Name AS AgencyName,
COUNT(sr.ServiceRequestId) AS CompletedRequests
FROM ServiceRequest sr
JOIN Manager m ON sr.ManagerId = m.ManagerId
JOIN Agency a ON m.AgencyId = a.AgencyId
WHERE sr.Status = 'Completed' AND sr.Date >= DATE_SUB(CURDATE(),
INTERVAL 1 MONTH)
GROUP BY m.ManagerId, m.ManagerName, a.Name;
```

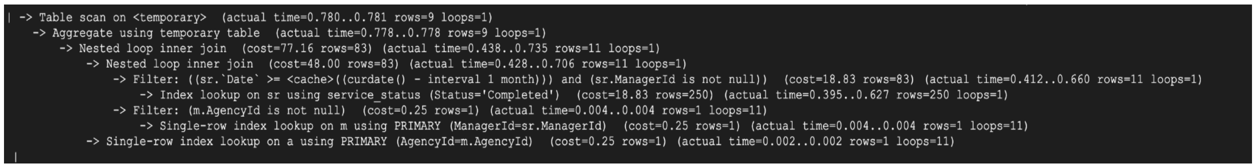
**Before Indexing:**



```
| -> Table scan on <temporary>  (actual time=0.848..0.850 rows=9 loops=1)
    -> Aggregate using temporary table  (actual time=0.848..0.848 rows=9 loops=1)
        -> Nested loop inner join  (cost=120.16 rows=33) (actual time=0.197..0.812 rows=11 loops=1)
            -> Nested loop inner join  (cost=108.50 rows=33) (actual time=0.190..0.788 rows=11 loops=1)
                -> Filter: ((sr.`Status` = 'Completed') and (sr.`Date` >= <cache>((curdate() - interval 1 month))) and (sr.ManagerId is not null))  (cost=96.83 rows=33) (actual time=0.129..0
.617 rows=11 loops=1)
                    -> Table scan on sr  (cost=96.83 rows=1000) (actual time=0.095..0.472 rows=1000 loops=1)
                -> Filter: (m.AgencyId is not null)  (cost=0.25 rows=1) (actual time=0.013..0.013 rows=1 loops=11)
                    -> Single-row index lookup on m using PRIMARY (ManagerId=sr.ManagerId)  (cost=0.25 rows=1) (actual time=0.013..0.013 rows=1 loops=11)
            -> Single-row index lookup on a using PRIMARY (AgencyId=m.AgencyId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=11)
 |
```

**After Indexing:**

1. **create index service_status on ServiceRequest(Status);**



```
| -> Table scan on <temporary>  (actual time=0.780..0.781 rows=9 loops=1)
    -> Aggregate using temporary table  (actual time=0.778..0.778 rows=9 loops=1)
        -> Nested loop inner join  (cost=77.16 rows=83) (actual time=0.438..0.735 rows=11 loops=1)
            -> Nested loop inner join  (cost=48.00 rows=83) (actual time=0.428..0.706 rows=11 loops=1)
                -> Filter: ((sr.`Date` >= <cache>((curdate() - interval 1 month))) and (sr.ManagerId is not null))  (cost=18.83 rows=83) (actual time=0.412..0.660 rows=11 loops=1)
                    -> Index lookup on sr using service_status (Status='Completed')  (cost=18.83 rows=250) (actual time=0.395..0.627 rows=250 loops=1)
                -> Filter: (m.AgencyId is not null)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=11)
                    -> Single-row index lookup on m using PRIMARY (ManagerId=sr.ManagerId)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=11)
            -> Single-row index lookup on a using PRIMARY (AgencyId=m.AgencyId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=11)
 |
```

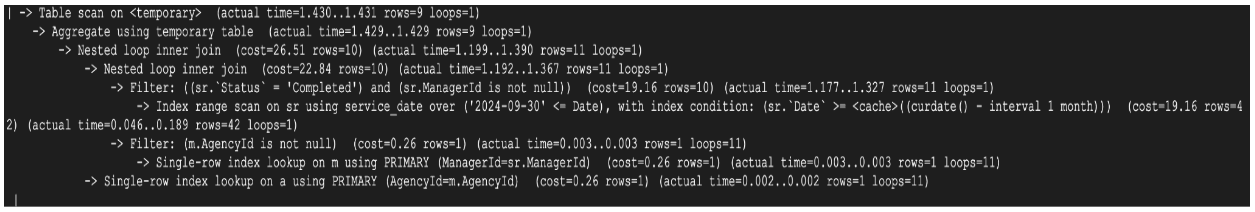
Cost reduced from

**120.16 -> 77.6**

**108.50 -> 48**

**96.83 -> 18.83**

2. **create index service_date on ServiceRequest(Date);**



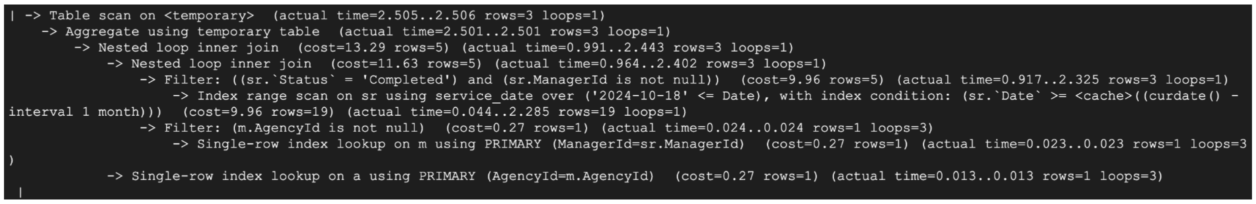
```
| -> Table scan on <temporary>  (actual time=1.430..1.431 rows=9 loops=1)
    -> Aggregate using temporary table  (actual time=1.429..1.429 rows=9 loops=1)
        -> Nested loop inner join  (cost=26.51 rows=10) (actual time=1.199..1.390 rows=11 loops=1)
            -> Nested loop inner join  (cost=22.84 rows=10) (actual time=1.192..1.367 rows=11 loops=1)
                -> Filter: ((sr.`Status` = 'Completed') and (sr.ManagerId is not null))  (cost=19.16 rows=10) (actual time=1.177..1.327 rows=11 loops=1)
                    -> Index range scan on sr using service_date over ('2024-09-30' <= Date), with index condition: (sr.`Date` >= <cache>((curdate() - interval 1 month)))  (cost=19.16 rows=4
2) (actual time=0.046..0.189 rows=42 loops=1)
                -> Filter: (m.AgencyId is not null)  (cost=0.26 rows=1) (actual time=0.003..0.003 rows=1 loops=11)
                    -> Single-row index lookup on m using PRIMARY (ManagerId=sr.ManagerId)  (cost=0.26 rows=1) (actual time=0.003..0.003 rows=1 loops=11)
            -> Single-row index lookup on a using PRIMARY (AgencyId=m.AgencyId)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=11)
 |
```

Cost reduced from

**77.6 -> 26.51**

**48 -> 22.84**

3. **Create index manager_name on Manager(ManagerName);**



```
| -> Table scan on <temporary>  (actual time=2.505..2.506 rows=3 loops=1)
    -> Aggregate using temporary table  (actual time=2.501..2.501 rows=3 loops=1)
        -> Nested loop inner join  (cost=13.29 rows=5) (actual time=0.991..2.443 rows=3 loops=1)
            -> Nested loop inner join  (cost=11.63 rows=5) (actual time=0.964..2.402 rows=3 loops=1)
                -> Filter: ((sr.`Status` = 'Completed') and (sr.ManagerId is not null))  (cost=9.96 rows=5) (actual time=0.917..2.325 rows=3 loops=1)
                    -> Index range scan on sr using service_date over ('2024-10-18' <= Date), with index condition: (sr.`Date` >= <cache>((curdate() -
interval 1 month)))  (cost=9.96 rows=19) (actual time=0.044..2.285 rows=19 loops=1)
                -> Filter: (m.AgencyId is not null)  (cost=0.27 rows=1) (actual time=0.024..0.024 rows=1 loops=3)
                    -> Single-row index lookup on m using PRIMARY (ManagerId=sr.ManagerId)  (cost=0.27 rows=1) (actual time=0.023..0.023 rows=1 loops=3
)
            -> Single-row index lookup on a using PRIMARY (AgencyId=m.AgencyId)  (cost=0.27 rows=1) (actual time=0.013..0.013 rows=1 loops=3)
 |
```

Cost reduced from

**26.51-> 13.29**

**22.84-> 11.63**

**Final index design:**

1. **create index service_status on ServiceRequest(Status);**

2. **`create index service_date on ServiceRequest(Date);`**
3. **`Create index manager_name on Manager(ManagerName);`**

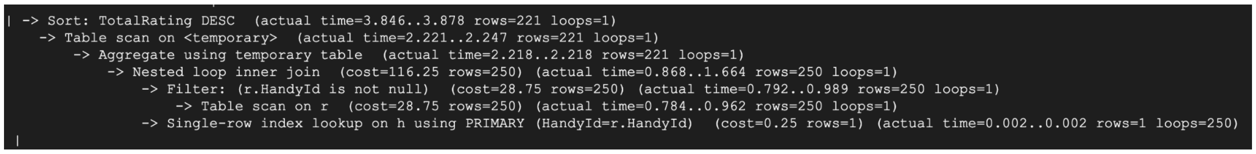
Creating an index on Status attribute used in the where clause, substantially reduced cost in all operations. Further using an index on Date attribute of ServiceRequest led to further optimization of the query leading to reduced cost as shown above. Indexing on the managerName attribute used in the groupby clause also helped further reduce the cost to the least minimum in all the tried combinations.

## Query 4: Retrieve Total Ratings and Number of Reviews for Each Handyperson, Grouped by Skill

This query groups handypersons by skill and calculates their total ratings and the number of reviews they've received, involving aggregation and a join.

```
SELECT h.Skills, h.Name AS HandypersonName, COUNT(r.ReviewId) AS
TotalReviews, SUM(r.Rating) AS TotalRating
FROM Handyperson h
JOIN Review r ON h.HandyId = r.HandyId
GROUP BY h.Skills, h.Name
ORDER BY TotalRating DESC;
```

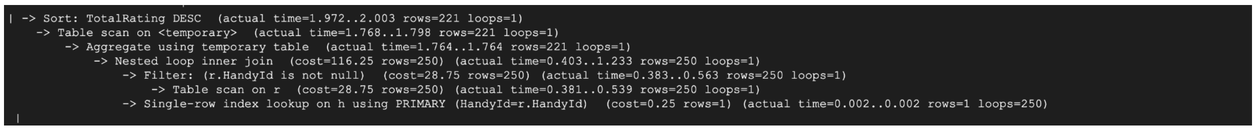
**Before Indexing:**



```
| -> Sort: TotalRating DESC  (actual time=3.846..3.878 rows=221 loops=1)
    -> Table scan on <temporary>  (actual time=2.221..2.247 rows=221 loops=1)
        -> Aggregate using temporary table  (actual time=2.218..2.218 rows=221 loops=1)
            -> Nested loop inner join  (cost=116.25 rows=250) (actual time=0.868..1.664 rows=250 loops=1)
                -> Filter: (r.HandyId is not null)  (cost=28.75 rows=250) (actual time=0.792..0.989 rows=250 loops=1)
                    -> Table scan on r  (cost=28.75 rows=250) (actual time=0.784..0.962 rows=250 loops=1)
                -> Single-row index lookup on h using PRIMARY (HandyId=r.HandyId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=250)
 |
```

**After Indexing:**

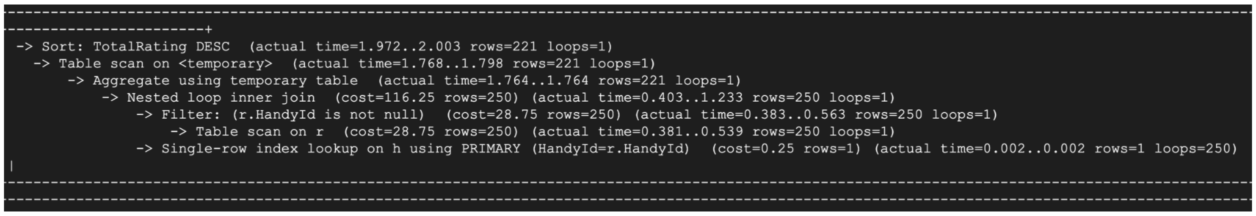
1. **`Create index handy_rating on Handyperson(Rating);`**



```
| -> Sort: TotalRating DESC  (actual time=1.972..2.003 rows=221 loops=1)
    -> Table scan on <temporary>  (actual time=1.768..1.798 rows=221 loops=1)
        -> Aggregate using temporary table  (actual time=1.764..1.764 rows=221 loops=1)
            -> Nested loop inner join  (cost=116.25 rows=250) (actual time=0.403..1.233 rows=250 loops=1)
                -> Filter: (r.HandyId is not null)  (cost=28.75 rows=250) (actual time=0.383..0.563 rows=250 loops=1)
                    -> Table scan on r  (cost=28.75 rows=250) (actual time=0.381..0.539 rows=250 loops=1)
                -> Single-row index lookup on h using PRIMARY (HandyId=r.HandyId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=250)
 |
```

No change in cost

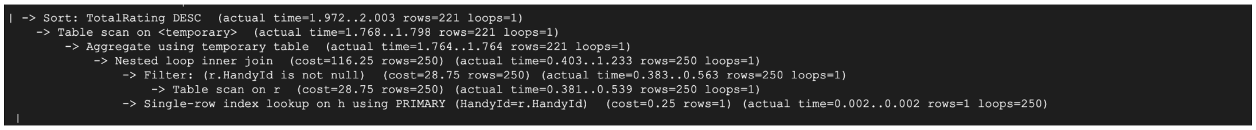
2. **create index handy_Names on Handyperson(Name);**



```
-----------------------+
 -> Sort: TotalRating DESC  (actual time=1.972..2.003 rows=221 loops=1)
   -> Table scan on <temporary>  (actual time=1.768..1.798 rows=221 loops=1)
       -> Aggregate using temporary table  (actual time=1.764..1.764 rows=221 loops=1)
           -> Nested loop inner join  (cost=116.25 rows=250) (actual time=0.403..1.233 rows=250 loops=1)
               -> Filter: (r.HandyId is not null)  (cost=28.75 rows=250) (actual time=0.383..0.563 rows=250 loops=1)
                   -> Table scan on r  (cost=28.75 rows=250) (actual time=0.381..0.539 rows=250 loops=1)
               -> Single-row index lookup on h using PRIMARY (HandyId=r.HandyId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=250)
|
```

No change in cost

3. **CREATE INDEX handy_skills_name ON Handyperson(Skills, Name);**



```
| -> Sort: TotalRating DESC  (actual time=1.972..2.003 rows=221 loops=1)
    -> Table scan on <temporary>  (actual time=1.768..1.798 rows=221 loops=1)
        -> Aggregate using temporary table  (actual time=1.764..1.764 rows=221 loops=1)
            -> Nested loop inner join  (cost=116.25 rows=250) (actual time=0.403..1.233 rows=250 loops=1)
                -> Filter: (r.HandyId is not null)  (cost=28.75 rows=250) (actual time=0.383..0.563 rows=250 loops=1)
                    -> Table scan on r  (cost=28.75 rows=250) (actual time=0.381..0.539 rows=250 loops=1)
                -> Single-row index lookup on h using PRIMARY (HandyId=r.HandyId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=250)
 |
```

No change in cost

**Final index design:**

It appears that in this case, the query plan might be too complicated for it to be optimised by indexing. The index might have had difficulty in dealing with the group by along with two aggregation methods, which lead to no overall cost optimization.