```
[1]+  Stopped                 gcloud sql connect sqldb411v1 --user=root --quiet
rithvik_kopparapu@cloudshell:~ (cs-411-charliebrown)$ gcloud sql connect sqldb411v1 --user=root --quiet
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 44
Server version: 8.4.0-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> use scentastic;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+----------------------+
| Tables_in_scentastic |
+----------------------+
| Dupes                |
| Perfumes             |
| Reviews              |
| Scent                |
| User                 |
+----------------------+
5 rows in set (0.00 sec)

mysql>
```

```
mysql> SELECT COUNT(*) FROM User;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> SELECT COUNT(*) FROM Perfumes;
+----------+
| COUNT(*) |
+----------+
|     1002 |
+----------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> SELECT COUNT(*) FROM Scent;
+----------+
| COUNT(*) |
+----------+
|     1002 |
+----------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> SELECT COUNT(*) FROM Reviews;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> SELECT COUNT(*) FROM Dupes;
+----------+
| COUNT(*) |
+----------+
|      996 |
+----------+
1 row in set (0.00 sec)
```

(some perfumes may not have dupes)

```
CREATE TABLE User (
   UserID VARCHAR(8) PRIMARY KEY,
   FirstName VARCHAR(255),
   LastName VARCHAR(255),
   Password VARCHAR(12),
   Notes VARCHAR(255),
   Feelings VARCHAR(255)
);

CREATE TABLE Perfumes (
   PerfumeID VARCHAR(8) PRIMARY KEY,
   Name VARCHAR(255),
   Price VARCHAR(255),
   Image VARCHAR(255),
   Brand VARCHAR(255)
);

CREATE TABLE Reviews (
   ReviewID VARCHAR(8) PRIMARY KEY,
   PerfumeID VARCHAR(8),
   ReviewerName VARCHAR(255),
   Notes VARCHAR(255),
   Feelings VARCHAR(255),
   FOREIGN KEY (PerfumeID) REFERENCES Perfumes(PerfumeID)
```

```
);

CREATE TABLE Scent (
   ScentID VARCHAR(8) PRIMARY KEY,
   PerfumeID VARCHAR(8),
   Notes VARCHAR(255),
   Feelings VARCHAR(255),
   FOREIGN KEY (PerfumeID) REFERENCES Perfumes(PerfumeID)
);

CREATE TABLE Dupes (
   DupeID VARCHAR(8) PRIMARY KEY,
   OriginalID VARCHAR(8),
   ogBrand VARCHAR(255),
   dBrand VARCHAR(255),
   FOREIGN KEY (OriginalID) REFERENCES Perfumes(PerfumeID)
);
```

Query 1 to Get Perfumes with Similar “Feelings” or “Notes” Descriptions as Dupes (Using Join and Set Operators)

This query retrieves perfumes that have dupes with similar Feelings or Notes based on the Scent table. It uses a join and a UNION set operation.

```
SELECT s1.PerfumeID, s1.Notes, s1.Feelings
FROM Scent s1
JOIN Dupes d ON s1.PerfumeID = d.OriginalID
WHERE EXISTS (
   SELECT 1
   FROM Scent s2
   WHERE s2.PerfumeID = d.DupeID
   AND (s2.Notes = s1.Notes OR s2.Feelings = s1.Feelings)
)
UNION
SELECT s2.PerfumeID, s2.Notes, s2.Feelings
FROM Scent s2
JOIN Dupes d ON s2.PerfumeID = d.DupeID
WHERE EXISTS (
   SELECT 1
   FROM Scent s1
   WHERE s1.PerfumeID = d.OriginalID
   AND (s1.Notes = s2.Notes OR s1.Feelings = s2.Feelings)
);
```

```
mysql> SELECT s1.PerfumeID, s1.Notes, s1.Feelings FROM Scent s1 JOIN Dupes d ON s1.PerfumeID = d.OriginalID WHERE EXISTS (     SELECT 1      FROM Scent s2      WHERE s2.PerfumeID = d.DupeID
   AND (s2.Notes = s1.Notes OR s2.Feelings = s1.Feelings) ) UNION SELECT s2.PerfumeID, s2.Notes, s2.Feelings FROM Scent s2 JOIN Dupes d ON s2.PerfumeID = d.DupeID WHERE EXISTS (     SELECT 1
    FROM Scent s1      WHERE s1.PerfumeID = d.OriginalID      AND (s1.Notes = s2.Notes OR s1.Feelings = s2.Feelings) )LIMIT 15;
+-----------+--------+--------------+
| PerfumeID | Notes  | Feelings     |
+-----------+--------+--------------+
| 952       | Sweet  | Refreshing   |
| 953       | Earthy | Warm         |
| 954       | Floral | Calming      |
| 955       | Fresh  | Energetic    |
| 956       | Earthy | Refreshing   |
| 957       | Floral | Sensual      |
| 958       | Citrus | Invigorating |
| 959       | Earthy | Bold         |
| 960       | Spicy  | Sensual      |
| 965       | Sweet  | Soft         |
| 966       | Fruity | Warm         |
| 967       | Fresh  | Soft         |
| 968       | Fruity | Refreshing   |
| 970       | Floral | Refreshing   |
| 971       | Sweet  | Warm         |
+-----------+--------+--------------+
15 rows in set, 6 warnings (0.01 sec)
```

Query 2 to List Each Perfume and the Average Price of Its Dupes (Aggregation and Join with a Subquery)

This query retrieves each perfume along with the average price of all its dupes by joining Perfumes and Dupes tables and aggregating.

```
SELECT p.Name AS PerfumeName, p.Brand,
      AVG(CAST(d_perf.Price AS DECIMAL(10, 2))) AS AvgDupePrice
FROM Perfumes p
JOIN Dupes d ON p.PerfumeID = d.OriginalID
JOIN Perfumes d_perf ON d.DupeID = d_perf.PerfumeID
GROUP BY p.PerfumeID, p.Name, p.Brand;
```

```
mysql> SELECT p.Name AS PerfumeName, p.Brand,         AVG(CAST(d_perf.Price AS DECIMAL(10, 2))) AS AvgDupePrice FROM Perfumes p JOIN Dupes d ON p.PerfumeID = d.OriginalID JOIN Perfumes d_perf
ON d.DupeID = d_perf.PerfumeID GROUP BY p.PerfumeID, p.Name, p.Brand LIMIT 15;
+-------------------------------------------------------------------------------------+--------------------+--------------+
| PerfumeName                                                                         | Brand              | AvgDupePrice |
+-------------------------------------------------------------------------------------+--------------------+--------------+
| Victoria's Secret Bombshell Escape Eau De Parfum 3.4oz                              | Victoria's Secret  |    30.820000 |
| Discontinued REALITIES By Liz Claiborne 3.4oz Eau De Parfum (True Photo)            | Liz Claiborne      |    29.389379 |
| I Love Love Cheap and Chic by Moschino EDT For Women 100 ml *NEW*                   | Moschino           |    17.118286 |
| Attar Al Wesal EDP Perfume By Al Wataniah 100 ML🦊Hot New Super Rich Fragrance🦊      | Al Wataniah        |    23.562143 |
| BURBERRY BODY EDP 2.0ml .06fl oz x 5 PERFUME SPRAY SAMPLE VIALS                     | Burberry           |     7.126667 |
| Elie Saab Girl of Now Shine EDP 1.0 oz Fragrances 7640233340233                     | Elie Saab          |    23.806154 |
| VALENTINO Perfume Travel Spray 10ml (Choose Your Scent) Combined Shipping           | Valentino          |    16.650938 |
| Sol De Janeiro Rio Perfume Mist 1fl.oz./30ml Travel Size, Individual Sprays         | Sol De Janeiro     |    13.702727 |
| L.A.M.B by Gwen Stefani LAMB perfume 0.17 oz / 5 ml EDP for Women                   | Gwen Stefani       |    21.826923 |
| JUICY COUTURE DUO: VIVA LA JUICY W EDP  / VIVA LA JUICY GOLD COUTURE W EDP (5ML)    | Juicy Couture      |     5.650000 |
| New YSL Mon Paris Eau De Parfum 3 oz/90ml Yves Saint Laurent EDP Spray for Women    | Yves Saint Laurent |    27.238846 |
| Versace Dylan Purple Women EDP 3.4 Oz Spray lotion Shower gel & Mini 4 Pc Set       | Versace            |    45.873196 |
| Narciso Rodriguez Musc Noir Rose 3.3 Oz /100ml Eau de Parfum for Women New Us       | As Show            |    42.358475 |
| VERSACE EROS POUR FEMME 3.4 oz 3.3 edt Perfume New Tester                           | Versace            |    40.600000 |
| PARFUMS de MARLY DELINA La Rosee 2.5 oz./ 75 ml. EDP Spray- NEW IN BOX              | Parfums de Marly   |    60.330233 |
+-------------------------------------------------------------------------------------+--------------------+--------------+
15 rows in set (0.00 sec)
```

Query 3: List Perfumes Reviewed Positively by Reviewers, Excluding Perfumes with Lower Ratings (Using Join and Set Operators)

This query finds perfumes that received only positive reviews, based on keywords in the Feelings column. It uses a join and set operation to exclude perfumes that also have negative reviews.

```
SELECT p.Name, p.Brand

FROM Perfumes p
```

```
JOIN Reviews r ON p.PerfumeID = r.PerfumeID

WHERE r.Feelings LIKE '%happy%' OR r.Feelings LIKE '%love%'

OR p.PerfumeID NOT IN (

   SELECT PerfumeID

   FROM Reviews

   WHERE Feelings LIKE '%dislike%' OR Feelings LIKE '%sad%'

)

LIMIT 15;
```

```
mysql> SELECT p.Name, p.Brand
    -> FROM Perfumes p
    -> JOIN Reviews r ON p.PerfumeID = r.PerfumeID
    -> WHERE r.Feelings LIKE '%happy%' OR r.Feelings LIKE '%love%'
    -> OR p.PerfumeID NOT IN (
    ->     SELECT PerfumeID
    ->     FROM Reviews
    ->     WHERE Feelings LIKE '%dislike%' OR Feelings LIKE '%sad%'
    -> )
    -> LIMIT 15;
+--------------------------------------------------------------------------------+------------------+
| Name                                                                           | Brand            |
+--------------------------------------------------------------------------------+------------------+
| PRADA Paradoxe by Prada EDP 3.0oz/90ml Spray Perfume for Women New In Box      | PRADA            |
| Lattafa YARA by Lattafa 3.4 Oz (100 ml) EDP Eau De Parfum Spray for Women.     | Carolina Herrera |
| Shiyaaka for Men EDP Spray 100ML (3.4 FL.OZ) By Khadlaj (Woody, Aromatic, Earth) | Khadlaj          |
| Amazing Grace by Philosophy, 4 oz EDP Intense Spray for Women                  | Carolina Herrera |
| Burberry Touch by Burberry 3.3 / 3.4 oz EDP Spray For Women Brand New Sealed   | Burberry         |
| AMOR AMOR by Cacharel Perfume for women 3.3 oz / 3.4 oz edt New                | Cacharel         |
| Yves Saint Laurent Black Opium 3oz Eau de Parfum Women's New Sealed            | Carolina Herrera |
| FLOWERBOMB BY VIKTOR & ROLF 3.4 OZ SPRAY EAU DE PARFUM SPRAY NEW & SEALED      | Viktor & Rolf    |
| Prada Les infusions De Milano Iris Cedre for Women 3.4oz/100ml EDP Spray In Box | PRADA            |
| Versace Dylan Turquoise by Gianni Versace for women EDT 3.3 / 3.4 oz New Tester | Versace          |
| Estee Lauder Cinnabar Eau De Parfum Spray, 1.7 oz / 50 ml Perfume, NWOB        | Estée Lauder     |
| GAP DREAM BODY MIST 8FL OZ/236 mL FREE SHIPPING NEW                            | Gap              |
| Gucci Flora Gorgeous Gardenia 3.3oz Women's Eau de Toilette Spray New Sealed Box | Gucci            |
| Coach Wild Rose 3.0 oz EDP eau de parfum spray womens perfume 90 ml Tester     | Coach            |
| Parfums de Marly Delina La Rosee Eau de Parfum 2.5 Fl Oz/75ml Spray for Women  | As Shown         |
+--------------------------------------------------------------------------------+------------------+
15 rows in set (0.00 sec)

mysql>
```

Query 4 to Find the Top 15 Most Reviewed Perfumes with Aggregation and Join

This query finds the top 15 perfumes with the highest number of reviews, joining the Perfumes and Reviews tables and aggregating with GROUP BY.

```
SELECT p.Name AS PerfumeName, p.Brand, COUNT(r.ReviewID) AS NumberOfReviews

FROM Perfumes p
```

JOIN Reviews r ON p.PerfumeID = r.PerfumeID

GROUP BY p.PerfumeID, p.Name, p.Brand

ORDER BY NumberOfReviews DESC

LIMIT 15;

```
mysql> SELECT p.Name AS PerfumeName, p.Brand, COUNT(r.ReviewID) AS NumberOfReviews
    -> FROM Perfumes p
    -> JOIN Reviews r ON p.PerfumeID = r.PerfumeID
    -> GROUP BY p.PerfumeID, p.Name, p.Brand
    -> ORDER BY NumberOfReviews DESC
    -> LIMIT 15;
+--------------------------------------------------------------------------------+--------------------+-----------------+
| PerfumeName                                                                    | Brand              | NumberOfReviews |
+--------------------------------------------------------------------------------+--------------------+-----------------+
| Versace Dylan Turquoise by Gianni Versace for women EDT 3.3 / 3.4 oz New Tester | Versace            |              18 |
| Elizabeth Arden White Tea Fragrance Collection Set for Women - 3Pc Mini Gift Set | Elizabeth Arden    |              16 |
| Valentine Milano For Women Perfume 3.4 fl.oz from Fragrance Couture.           | Valentino          |              15 |
| Women Perfume 3.4 oz / 100 ml Eau De Toilette Spray                            | AS  SHOWN          |              15 |
| Marc Jacobs Daisy Women's Eau De Toilette Spray For Women EDT 3.4Oz 100ml New  | Marc Jacobs        |              15 |
| Oxytocin-N 15iu Per Spray 1oz (30ml)                                           | Carolina Herrera   |              15 |
| Not a Perfume by Juliette Has A Gun, 3.3 oz EDP Spray for Women Eau De Parfum  | Juliette has a gun |              14 |
| Mon Paris 3 oz Perfume by Yves Saint Laurent 90 ml Womens Spray EDP New & Sealed | Yves Saint Laurent |              14 |
| Urban Outfitters Le Monde  Gourmand Eau De Parfum Fraise Fouettee 070 1oz.      | Urban Outfitters   |              14 |
| Thierry Mugler Angel 1.7oz EDP Women's Perfume Spray New Sealed                | Thierry Mugler     |              14 |
| Love by Kilian Don't Be Shy Eau de Parfum 7.5 ml/0.25 fl. oz. Free Shipping …  | Kilian             |              14 |
| Carolina Herrera Good Girl 2.7 oz Eau De Parfum Spray Women's New & Sealed     | Carolina Herrera   |              13 |
| My Way by Giorgio Armani 3 oz EDP Perfume for Women New In Box                 | Giorgio² Armani    |              13 |
| FLOWERBOMB BY VIKTOR & ROLF 3.4 OZ SPRAY EAU DE PARFUM SPRAY NEW & SEALED      | Viktor & Rolf      |              13 |
| Burberry Touch by Burberry 3.3 / 3.4 oz EDP Spray For Women Brand New Sealed   | Burberry           |              13 |
+--------------------------------------------------------------------------------+--------------------+-----------------+
15 rows in set (0.01 sec)

mysql>
```

QUERY 1:

Before Indexing:

```
mysql> EXPLAIN ANALYZE SELECT s1.PerfumeID, s1.Notes, s1.Feelings FROM Scent s1 JOIN Dupes d ON s1.PerfumeID = d.OriginalID WHERE EXISTS (SELECT 1 FROM Scent s2 WHERE s2.PerfumeID = d.DupeID
AND (s2.Notes = s1.Notes OR s2.Feelings = s1.Feelings)) UNION SELECT s2.PerfumeID, s2.Notes, s2.Feelings FROM Scent s2 JOIN Dupes d ON s2.PerfumeID = d.DupeID WHERE EXISTS (SELECT 1 FROM Scent
 s1 WHERE s1.PerfumeID = d.OriginalID AND (s1.Notes = s2.Notes OR s1.Feelings = s2.Feelings));
```

```
| -> Table scan on <union temporary>  (cost=1541..1548 rows=354) (actual time=15..15.1 rows=938 loops=1)
    -> Union materialize with deduplication  (cost=1541..1541 rows=354) (actual time=15..15 rows=938 loops=1)
        -> Nested loop semijoin  (cost=753 rows=177) (actual time=0.0407..7.14 rows=932 loops=1)
            -> Nested loop inner join  (cost=426 rows=932) (actual time=0.0324..3.73 rows=932 loops=1)
                -> Filter: (d.OriginalID is not null)  (cost=100 rows=932) (actual time=0.0159..0.241 rows=932 loops=1)
                    -> Covering index scan on d using idx_dupe_og_id  (cost=100 rows=932) (actual time=0.0154..0.18 rows=932 loops=1)
                -> Index lookup on s1 using PerfumeID (PerfumeID=d.OriginalID)  (cost=0.25 rows=1) (actual time=0.00322..0.00362 rows=1 loops=932)
            -> Filter: ((s2.Notes = s1.Notes) or (s2.Feelings = s1.Feelings))  (cost=0.0475 rows=0.19) (actual time=0.00356..0.00356 rows=1 loops=932)
                -> Index lookup on s2 using PerfumeID (PerfumeID=d.DupeID)  (cost=0.0475 rows=1) (actual time=0.00329..0.00329 rows=1 loops=932)
        -> Nested loop semijoin  (cost=753 rows=177) (actual time=0.0201..7.03 rows=932 loops=1)
    -> Union materialize with deduplication  (cost=1541..1541 rows=354) (actual time=15..15 rows=938 loops=1)
        -> Nested loop semijoin  (cost=753 rows=177) (actual time=0.0407..7.14 rows=932 loops=1)
            -> Nested loop inner join  (cost=426 rows=932) (actual time=0.0324..3.73 rows=932 loops=1)
                -> Filter: (d.OriginalID is not null)  (cost=100 rows=932) (actual time=0.0159..0.241 rows=932 loops=1)
                    -> Covering index scan on d using idx_dupe_og_id  (cost=100 rows=932) (actual time=0.0154..0.18 rows=932 loops=1)
                -> Index lookup on s1 using PerfumeID (PerfumeID=d.OriginalID)  (cost=0.25 rows=1) (actual time=0.00322..0.00362 rows=1 loops=932)
            -> Filter: ((s2.Notes = s1.Notes) or (s2.Feelings = s1.Feelings))  (cost=0.0475 rows=0.19) (actual time=0.00356..0.00356 rows=1 loops=932)
                -> Index lookup on s2 using PerfumeID (PerfumeID=d.DupeID)  (cost=0.0475 rows=1) (actual time=0.00329..0.00329 rows=1 loops=932)
        -> Nested loop semijoin  (cost=753 rows=177) (actual time=0.0201..7.03 rows=932 loops=1)
            -> Nested loop inner join  (cost=426 rows=932) (actual time=0.0153..3.77 rows=932 loops=1)
                -> Filter: (d.OriginalID is not null)  (cost=100 rows=932) (actual time=0.0101..0.239 rows=932 loops=1)
                    -> Covering index scan on d using idx_dupe_og_id  (cost=100 rows=932) (actual time=0.00989..0.173 rows=932 loops=1)
                -> Index lookup on s2 using PerfumeID (PerfumeID=d.DupeID)  (cost=0.25 rows=1) (actual time=0.00328..0.00367 rows=1 loops=932)
            -> Filter: ((s1.Notes = s2.Notes) or (s1.Feelings = s2.Feelings))  (cost=0.0475 rows=0.19) (actual time=0.00341..0.00341 rows=1 loops=932)
                -> Index lookup on s1 using PerfumeID (PerfumeID=d.OriginalID)  (cost=0.0475 rows=1) (actual time=0.00315..0.00315 rows=1 loops=932)
 |
```

After “CREATE INDEX idx_scent_feelings ON Scent(Feelings);”

```
mysql> EXPLAIN ANALYZE SELECT s1.PerfumeID, s1.Notes, s1.Feelings FROM Scent s1 JOIN Dupes d ON s1.PerfumeID = d.OriginalID WHERE EXISTS (SELECT 1 FROM Scent s2 WHERE s2.PerfumeID = d.DupeID
AND (s2.Notes = s1.Notes OR s2.Feelings = s1.Feelings)) UNION SELECT s2.PerfumeID, s2.Notes, s2.Feelings FROM Scent s2 JOIN Dupes d ON s2.PerfumeID = d.DupeID WHERE EXISTS (SELECT 1 FROM Scent
 s1 WHERE s1.PerfumeID = d.OriginalID AND (s1.Notes = s2.Notes OR s1.Feelings = s2.Feelings));
```

```
| -> Table scan on <union temporary>  (cost=1545..1552 rows=396) (actual time=15..15.1 rows=938 loops=1)
    -> Union materialize with deduplication  (cost=1545..1545 rows=396) (actual time=15..15 rows=938 loops=1)
        -> Nested loop semijoin  (cost=753 rows=198) (actual time=0.0423..7.15 rows=932 loops=1)
            -> Nested loop inner join  (cost=426 rows=932) (actual time=0.0344..3.73 rows=932 loops=1)
                -> Filter: (d.OriginalID is not null)  (cost=100 rows=932) (actual time=0.0168..0.242 rows=932 loops=1)
                    -> Covering index scan on d using idx_dupe_og_id  (cost=100 rows=932) (actual time=0.0162..0.186 rows=932 loops=1)
                -> Index lookup on s1 using PerfumeID (PerfumeID=d.OriginalID)  (cost=0.25 rows=1) (actual time=0.00321..0.00362 rows=1 loops=932)
            -> Filter: ((s2.Notes = s1.Notes) or (s2.Feelings = s1.Feelings))  (cost=0.0531 rows=0.212) (actual time=0.00358..0.00358 rows=1 loops=932)
                -> Index lookup on s2 using PerfumeID (PerfumeID=d.DupeID)  (cost=0.0531 rows=1) (actual time=0.00332..0.00332 rows=1 loops=932)
        -> Nested loop semijoin  (cost=753 rows=198) (actual time=0.0255..7.01 rows=932 loops=1)
            -> Nested loop inner join  (cost=426 rows=932) (actual time=0.0193..3.77 rows=932 loops=1)
                -> Filter: (d.OriginalID is not null)  (cost=100 rows=932) (actual time=0.0125..0.236 rows=932 loops=1)
                    -> Covering index scan on d using idx_dupe_og_id  (cost=100 rows=932) (actual time=0.0121..0.174 rows=932 loops=1)
                -> Index lookup on s2 using PerfumeID (PerfumeID=d.DupeID)  (cost=0.25 rows=1) (actual time=0.00328..0.00367 rows=1 loops=932)
            -> Filter: ((s1.Notes = s2.Notes) or (s1.Feelings = s2.Feelings))  (cost=0.0531 rows=0.212) (actual time=0.00338..0.00338 rows=1 loops=932)
                -> Index lookup on s1 using PerfumeID (PerfumeID=d.OriginalID)  (cost=0.0531 rows=1) (actual time=0.00312..0.00312 rows=1 loops=932)
 |
```

We thought that this would benefit the cost of the query because of its use in the where clause, but potentially the index requires more lookups within the subquery which could make it more expensive. This would be because the Feelings column has many of the same values, as perfumes overlap in the feelings they produce, which means that there is a low cardinality, and possibly slower lookups.

After “CREATE INDEX idx_scent_notes ON Scent(Notes);”

```
mysql> EXPLAIN ANALYZE SELECT s1.PerfumeID, s1.Notes, s1.Feelings FROM Scent s1 JOIN Dupes d ON s1.PerfumeID = d.OriginalID WHERE EXISTS (SELECT 1 FROM Scent s2 WHERE s2.PerfumeID = d.DupeID
AND (s2.Notes = s1.Notes OR s2.Feelings = s1.Feelings)) UNION SELECT s2.PerfumeID, s2.Notes, s2.Feelings FROM Scent s2 JOIN Dupes d ON s2.PerfumeID = d.DupeID WHERE EXISTS (SELECT 1 FROM Scent
 s1 WHERE s1.PerfumeID = d.OriginalID AND (s1.Notes = s2.Notes OR s1.Feelings = s2.Feelings));
```

```
| -> Table scan on <union temporary>  (cost=1549..1557 rows=437) (actual time=15..15.1 rows=938 loops=1)
    -> Union materialize with deduplication  (cost=1549..1549 rows=437) (actual time=15..15 rows=938 loops=1)
        -> Nested loop semijoin  (cost=753 rows=218) (actual time=0.0622..7.14 rows=932 loops=1)
            -> Nested loop inner join  (cost=426 rows=932) (actual time=0.0462..3.72 rows=932 loops=1)
                -> Filter: (d.OriginalID is not null)  (cost=100 rows=932) (actual time=0.0253..0.247 rows=932 loops=1)
                    -> Covering index scan on d using idx_dupe_og_id  (cost=100 rows=932) (actual time=0.0246..0.194 rows=932 loops=1)
                -> Index lookup on s1 using PerfumeID (PerfumeID=d.OriginalID)  (cost=0.25 rows=1) (actual time=0.00319..0.00361 rows=1 loops=932)
            -> Filter: ((s2.Notes = s1.Notes) or (s2.Feelings = s1.Feelings))  (cost=0.0586 rows=0.234) (actual time=0.00357..0.00357 rows=1 loops=932)
                -> Index lookup on s2 using PerfumeID (PerfumeID=d.DupeID)  (cost=0.0586 rows=1) (actual time=0.0033..0.0033 rows=1 loops=932)
        -> Nested loop semijoin  (cost=753 rows=218) (actual time=0.0237..7.02 rows=932 loops=1)
            -> Nested loop inner join  (cost=426 rows=932) (actual time=0.0186..3.79 rows=932 loops=1)
                -> Filter: (d.OriginalID is not null)  (cost=100 rows=932) (actual time=0.0112..0.25 rows=932 loops=1)
                    -> Covering index scan on d using idx_dupe_og_id  (cost=100 rows=932) (actual time=0.0109..0.178 rows=932 loops=1)
                -> Index lookup on s2 using PerfumeID (PerfumeID=d.DupeID)  (cost=0.25 rows=1) (actual time=0.00329..0.00368 rows=1 loops=932)
            -> Filter: ((s1.Notes = s2.Notes) or (s1.Feelings = s2.Feelings))  (cost=0.0586 rows=0.234) (actual time=0.00337..0.00337 rows=1 loops=932)
                -> Index lookup on s1 using PerfumeID (PerfumeID=d.OriginalID)  (cost=0.0586 rows=1) (actual time=0.00311..0.00311 rows=1 loops=932)
```

This increased the cost by quite a bit, having the table scan cost almost by 3 times. This was surprising to us, but the increase in cost may be to do with the fact that we are performing a

union between 2 subqueries and utilizing this index may increase cost because it's looking up the same thing multiple times.

After "CREATE INDEX idx_dupes_og_id ON Dupes(OriginalID);"

```
mysql> EXPLAIN ANALYZE SELECT s1.PerfumeID, s1.Notes, s1.Feelings FROM Scent s1 JOIN Dupes d ON s1.PerfumeID = d.OriginalID WHERE EXISTS (SELECT 1 FROM Scent s2 WHERE s2.PerfumeID = d.DupeID
AND (s2.Notes = s1.Notes OR s2.Feelings = s1.Feelings)) UNION SELECT s2.PerfumeID, s2.Notes, s2.Feelings FROM Scent s2 JOIN Dupes d ON s2.PerfumeID = d.DupeID WHERE EXISTS (SELECT 1 FROM Scent
 s1 WHERE s1.PerfumeID = d.OriginalID AND (s1.Notes = s2.Notes OR s1.Feelings = s2.Feelings));
```

```
| -> Table scan on <union temporary>  (cost=1541..1548 rows=354) (actual time=15..15.1 rows=938 loops=1)
    -> Union materialize with deduplication  (cost=1541..1541 rows=354) (actual time=15..15 rows=938 loops=1)
        -> Nested loop semijoin  (cost=753 rows=177) (actual time=0.0407..7.14 rows=932 loops=1)
            -> Nested loop inner join  (cost=426 rows=932) (actual time=0.0324..3.73 rows=932 loops=1)
                -> Filter: (d.OriginalID is not null)  (cost=100 rows=932) (actual time=0.0159..0.241 rows=932 loops=1)
                    -> Covering index scan on d using idx_dupe_og_id  (cost=100 rows=932) (actual time=0.0154..0.18 rows=932 loops=1)
                -> Index lookup on s1 using PerfumeID (PerfumeID=d.OriginalID)  (cost=0.25 rows=1) (actual time=0.00322..0.00362 rows=1 loops=932)
            -> Filter: ((s2.Notes = s1.Notes) or (s2.Feelings = s1.Feelings))  (cost=0.0475 rows=0.19) (actual time=0.00356..0.00356 rows=1 loops=932)
                -> Index lookup on s2 using PerfumeID (PerfumeID=d.DupeID)  (cost=0.0475 rows=1) (actual time=0.00329..0.00329 rows=1 loops=932)
        -> Nested loop semijoin  (cost=753 rows=177) (actual time=0.0201..7.03 rows=932 loops=1)
    -> Union materialize with deduplication  (cost=1541..1541 rows=354) (actual time=15..15 rows=938 loops=1)
        -> Nested loop semijoin  (cost=753 rows=177) (actual time=0.0407..7.14 rows=932 loops=1)
            -> Nested loop inner join  (cost=426 rows=932) (actual time=0.0324..3.73 rows=932 loops=1)
                -> Filter: (d.OriginalID is not null)  (cost=100 rows=932) (actual time=0.0159..0.241 rows=932 loops=1)
                    -> Covering index scan on d using idx_dupe_og_id  (cost=100 rows=932) (actual time=0.0154..0.18 rows=932 loops=1)
                -> Index lookup on s1 using PerfumeID (PerfumeID=d.OriginalID)  (cost=0.25 rows=1) (actual time=0.00322..0.00362 rows=1 loops=932)
            -> Filter: ((s2.Notes = s1.Notes) or (s2.Feelings = s1.Feelings))  (cost=0.0475 rows=0.19) (actual time=0.00356..0.00356 rows=1 loops=932)
                -> Index lookup on s2 using PerfumeID (PerfumeID=d.DupeID)  (cost=0.0475 rows=1) (actual time=0.00329..0.00329 rows=1 loops=932)
        -> Nested loop semijoin  (cost=753 rows=177) (actual time=0.0201..7.03 rows=932 loops=1)
            -> Nested loop inner join  (cost=426 rows=932) (actual time=0.0153..3.77 rows=932 loops=1)
                -> Filter: (d.OriginalID is not null)  (cost=100 rows=932) (actual time=0.0101..0.239 rows=932 loops=1)
                    -> Covering index scan on d using idx_dupe_og_id  (cost=100 rows=932) (actual time=0.00989..0.173 rows=932 loops=1)
                -> Index lookup on s2 using PerfumeID (PerfumeID=d.DupeID)  (cost=0.25 rows=1) (actual time=0.00328..0.00367 rows=1 loops=932)
            -> Filter: ((s1.Notes = s2.Notes) or (s1.Feelings = s2.Feelings))  (cost=0.0475 rows=0.19) (actual time=0.00341..0.00341 rows=1 loops=932)
                -> Index lookup on s1 using PerfumeID (PerfumeID=d.OriginalID)  (cost=0.0475 rows=1) (actual time=0.00315..0.00315 rows=1 loops=932)
 |
```

This indexing doesn't affect the cost due to the fact that this index is created with a foreign key which tends to already be indexed within the database structure. Although it's a foreign key, we hoped for some optimization to result as the OriginalID is referenced in the WHERE portion of the query. This addition did not affect cost, processing rows in the exact same way as before.

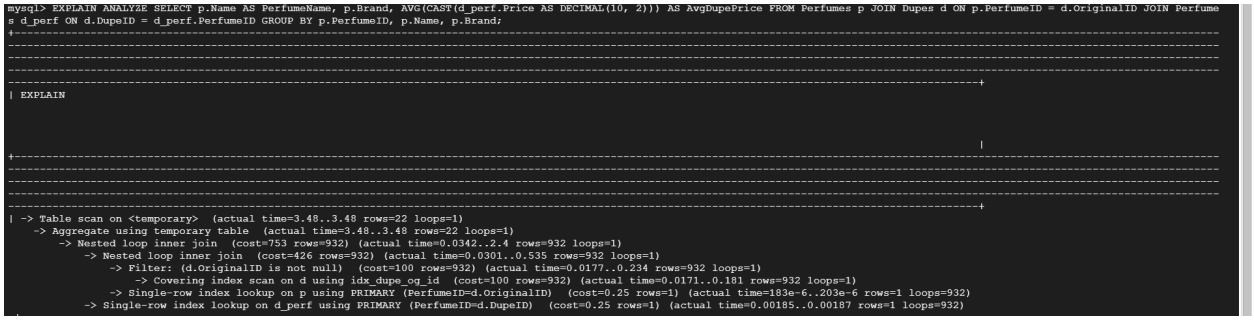
Upon attempting to optimize this query, we found that all of our methods of indexing yielded either the same results or worse results. Because of this, we decided to minimize any indexing that could be used for this query to simplify our database without losing any efficiency.

INDEXING QUERY 2:

Before Indexing:

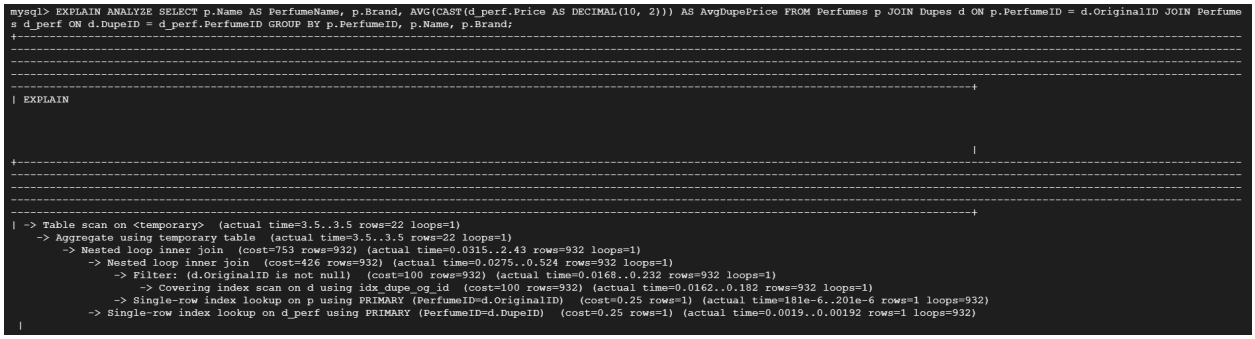
```
mysql> EXPLAIN ANALYZE SELECT p.Name AS PerfumeName, p.Brand, AVG(CAST(d_perf.Price AS DECIMAL(10,2))) AS AvgDupePrice FROM Perfumes p JOIN Dupes d ON p.PerfumeID = d.OriginalID JOIN Perfumes
 d_perf ON d.DupeID = d_perf.PerfumeID GROUP BY p.PerfumeID, p.Name, p.Brand;
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                                                                                 |
                                                                                                                                                                                          |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=3.46..3.47 rows=22 loops=1)
    -> Aggregate using temporary table  (actual time=3.46..3.46 rows=22 loops=1)
        -> Nested loop inner join  (cost=753 rows=932) (actual time=0.0448..2.42 rows=932 loops=1)
            -> Nested loop inner join  (cost=426 rows=932) (actual time=0.0403..0.552 rows=932 loops=1)
                -> Filter: (d.OriginalID is not null)  (cost=100 rows=932) (actual time=0.024..0.241 rows=932 loops=1)
                    -> Covering index scan on d using idx_dupe_og_id  (cost=100 rows=932) (actual time=0.0234..0.188 rows=932 loops=1)
                -> Single-row index lookup on p using PRIMARY (PerfumeID=d.OriginalID)  (cost=0.25 rows=1) (actual time=188e-6..208e-6 rows=1 loops=932)
            -> Single-row index lookup on d_perf using PRIMARY (PerfumeID=d.DupeID)  (cost=0.25 rows=1) (actual time=0.00185..0.00187 rows=1 loops=932)
```

After "CREATE INDEX idx_perfume_name ON Perfumes (name);" (same result)



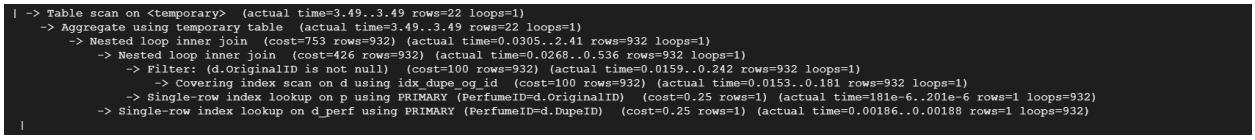
Indexing on perfumes name doesn't make a difference in the cost. This is due to the fact that this query is aggregating with PerfumeId, OriginalId, and DupeId of the dupes and perfumes, and not explicitly relying on the perfume name to do so. Although the perfume name is being utilized in the GroupBy, this doesn't optimize the cost at all.

After "CREATE INDEX idx_perfume_brand ON Perfumes(Brand);"



Indexing on perfume brand doesn't make a difference in the cost either for what we think is a very similar reason to the previous index of perfume name. Since the query is aggregating with PerfumeId, OriginalId, and DupeId of the dupes and perfumes, and not explicitly relying on the perfume brand to do so, it makes sense that it doesn't optimize the cost at all.

After "CREATE INDEX idx_dupe_og_id ON Dupes(OriginalID);"



This indexing of the OriginalID doesn't impact the cost either due to the fact that OriginalId in Dupes table is a foreign key which tends to already be indexed within the database. Although it's a foreign key, we hoped for some optimization to result as the OriginalID is referenced in the JOIN portion of the query.

After trying these indexing options for this query, we determined that we shouldn't include any extra indexing outside of what the database already has because it has no impact

on the efficiency of the query, which could make sense because this query doesn't have many components.

QUERY 3 INDEXING:

<u>Before Indexing:</u>

```
mysql> EXPLAIN ANALYZE SELECT p.Name, p.Brand FROM Perfumes p JOIN Reviews r ON p.PerfumeID = r.PerfumeID WHERE r.Feelings LIKE '%happy%' OR r.Feelings LIKE '%love%' OR p.PerfumeID NOT IN (
SELECT PerfumeID  FROM Reviews  WHERE Feelings LIKE '%dislike%' OR Feelings LIKE '%sad%');
+--------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                        |
+--------------------------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=458 rows=999) (actual time=16.1..18.9 rows=1000 loops=1)
    -> Filter: (r.PerfumeID is not null)  (cost=108 rows=999) (actual time=8.15..8.45 rows=1000 loops=1)
        -> Table scan on r  (cost=108 rows=999) (actual time=8.15..8.39 rows=1000 loops=1)
    -> Filter: ((r.Feelings like '%happy%') or (r.Feelings like '%love%') or <in_optimizer>(p.PerfumeID,p.PerfumeID in (select #2) is false))  (cost=0.25 rows=1) (actual time=0.0103..0.0103 ro
ws=1 loops=1000)
        -> Single-row index lookup on p using PRIMARY (PerfumeID=r.PerfumeID)  (cost=0.25 rows=1) (actual time=0.00186..0.00188 rows=1 loops=1000)
        -> Select #2 (subquery in condition; run only once)
            -> Filter: ((p.PerfumeID = `<materialized_subquery>`.PerfumeID))  (cost=129..129 rows=1) (actual time=7.95..7.95 rows=0 loops=1)
                -> Limit: 1 row(s)  (cost=129..129 rows=1) (actual time=7.95..7.95 rows=0 loops=1)
                    -> Index lookup on <materialized_subquery> using <auto_distinct_key> (PerfumeID=p.PerfumeID)  (actual time=7.95..7.95 rows=0 loops=1)
                        -> Materialize with deduplication  (cost=129..129 rows=210) (actual time=7.95..7.95 rows=0 loops=1)
                            -> Filter: ((Reviews.Feelings like '%dislike%') or (Reviews.Feelings like '%sad%'))  (cost=108 rows=210) (actual time=7.95..7.95 rows=0 loops=1)
                                -> Table scan on Reviews  (cost=108 rows=999) (actual time=0.0247..7.76 rows=1000 loops=1)
```

<u>After "CREATE INDEX idx_review_feelings ON Reviews(Feelings);"</u>

```
mysql> EXPLAIN ANALYZE SELECT p.Name, p.Brand FROM Perfumes p JOIN Reviews r ON p.PerfumeID = r.PerfumeID WHERE r.Feelings LIKE '%happy%' OR r.Feelings LIKE '%love%' OR p.PerfumeID NOT IN (
SELECT PerfumeID  FROM Reviews  WHERE Feelings LIKE '%dislike%' OR Feelings LIKE '%sad%');
+--------------------------------------------------------------------------------------------------------------------------------
--+
| EXPLAIN                                                                                                                        |
+--------------------------------------------------------------------------------------------------------------------------------
--+
| -> Nested loop inner join  (cost=452 rows=999) (actual time=0.446..3.19 rows=1000 loops=1)
    -> Filter: (r.PerfumeID is not null)  (cost=102 rows=999) (actual time=0.0334..0.325 rows=1000 loops=1)
        -> Table scan on r  (cost=102 rows=999) (actual time=0.0327..0.273 rows=1000 loops=1)
    -> Filter: ((r.Feelings like '%happy%') or (r.Feelings like '%love%') or <in_optimizer>(p.PerfumeID,p.PerfumeID in (select #2) is false))  (cost=0.25 rows=1) (actual time=0.00269..0.00275
rows=1 loops=1000)
        -> Single-row index lookup on p using PRIMARY (PerfumeID=r.PerfumeID)  (cost=0.25 rows=1) (actual time=0.00186..0.00188 rows=1 loops=1000)
        -> Select #2 (subquery in condition; run only once)
            -> Filter: ((p.PerfumeID = `<materialized_subquery>`.PerfumeID))  (cost=123..123 rows=1) (actual time=0.395..0.395 rows=0 loops=1)
                -> Limit: 1 row(s)  (cost=123..123 rows=1) (actual time=0.395..0.395 rows=0 loops=1)
                    -> Index lookup on <materialized_subquery> using <auto_distinct_key> (PerfumeID=p.PerfumeID)  (actual time=0.394..0.394 rows=0 loops=1)
                        -> Materialize with deduplication  (cost=123..123 rows=210) (actual time=0.394..0.394 rows=0 loops=1)
                            -> Filter: ((Reviews.Feelings like '%dislike%') or (Reviews.Feelings like '%sad%'))  (cost=102 rows=210) (actual time=0.392..0.392 rows=0 loops=1)
                                -> Table scan on Reviews  (cost=102 rows=999) (actual time=0.013..0.194 rows=1000 loops=1)
 |
```

This query benefitted from an index on Review Feelings. This is because it looks for what feelings are *not* in a selection, so by indexing by this, it may be able to narrow down the rows to then search for the "like" feelings part in. This "like" portion could also be benefited by this indexing because it can filter for what it should be like quicker.

<u>After "CREATE INDEX idx_perfume_name ON Perfumes(Names);"</u>

```
mysql> EXPLAIN ANALYZE SELECT p.Name, p.Brand FROM Perfumes p JOIN Reviews r ON p.PerfumeID = r.PerfumeID WHERE r.Feelings LIKE '%happy%' OR r.Feelings LIKE '%love%' OR p.PerfumeID NOT IN (
SELECT PerfumeID  FROM Reviews  WHERE Feelings LIKE '%dislike%' OR Feelings LIKE '%sad%');
+------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--+
| EXPLAIN



                                                                          |
+------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--+
| -> Nested loop inner join  (cost=452 rows=999) (actual time=0.434..3.14 rows=1000 loops=1)
    -> Filter: (r.PerfumeID is not null)  (cost=102 rows=999) (actual time=0.0283..0.313 rows=1000 loops=1)
        -> Table scan on r  (cost=102 rows=999) (actual time=0.0277..0.26 rows=1000 loops=1)
    -> Filter: ((r.Feelings like '%happy%') or (r.Feelings like '%love%') or <in_optimizer>(p.PerfumeID,p.PerfumeID in (select #2) is false))  (cost=0.25 rows=1) (actual time=0.00265..0.00271 
rows=1 loops=1000)
        -> Single-row index lookup on p using PRIMARY (PerfumeID=r.PerfumeID)  (cost=0.25 rows=1) (actual time=0.00184..0.00186 rows=1 loops=1000)
        -> Select #2 (subquery in condition; run only once)
            -> Filter: ((p.PerfumeID = `<materialized_subquery>`.PerfumeID))  (cost=123..123 rows=1) (actual time=0.389..0.389 rows=0 loops=1)
                -> Limit: 1 row(s)  (cost=123..123 rows=1) (actual time=0.389..0.389 rows=0 loops=1)
                    -> Index lookup on <materialized_subquery> using <auto_distinct_key> (PerfumeID=p.PerfumeID)  (actual time=0.389..0.389 rows=0 loops=1)
                        -> Materialize with deduplication  (cost=123..123 rows=210) (actual time=0.388..0.388 rows=0 loops=1)
                            -> Filter: ((Reviews.Feelings like '%dislike%') or (Reviews.Feelings like '%sad%'))  (cost=102 rows=210) (actual time=0.386..0.386 rows=0 loops=1)
                                -> Table scan on Reviews  (cost=102 rows=999) (actual time=0.0114..0.201 rows=1000 loops=1)
 |
```

Indexing by the Perfume name doesn't impact the cost of the query. We this think is because it isn't a part of the most expensive parts of the query like the join or the where, and only part of the select, which could make it unimpactful to the cost.

<u>After "CREATE INDEX idx_review_perfume ON Reviews(PerfumeID);"</u>

```
mysql> EXPLAIN ANALYZE SELECT p.Name, p.Brand FROM Perfumes p JOIN Reviews r ON p.PerfumeID = r.PerfumeID WHERE r.Feelings LIKE '%happy%' OR r.Feelings LIKE '%love%' OR p.PerfumeID NOT IN (
SELECT PerfumeID  FROM Reviews  WHERE Feelings LIKE '%dislike%' OR Feelings LIKE '%sad%');
+------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
---+
| EXPLAIN



                                                                          |
+------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
---+
| -> Nested loop inner join  (cost=452 rows=999) (actual time=0.455..3.19 rows=1000 loops=1)
    -> Filter: (r.PerfumeID is not null)  (cost=102 rows=999) (actual time=0.0329..0.334 rows=1000 loops=1)
        -> Table scan on r  (cost=102 rows=999) (actual time=0.0324..0.281 rows=1000 loops=1)
    -> Filter: ((r.Feelings like '%happy%') or (r.Feelings like '%love%') or <in_optimizer>(p.PerfumeID,p.PerfumeID in (select #2) is false))  (cost=0.25 rows=1) (actual time=0.00268..0.00274 
rows=1 loops=1000)
        -> Single-row index lookup on p using PRIMARY (PerfumeID=r.PerfumeID)  (cost=0.25 rows=1) (actual time=0.00185..0.00187 rows=1 loops=1000)
        -> Select #2 (subquery in condition; run only once)
            -> Filter: ((p.PerfumeID = `<materialized_subquery>`.PerfumeID))  (cost=123..123 rows=1) (actual time=0.399..0.399 rows=0 loops=1)
                -> Limit: 1 row(s)  (cost=123..123 rows=1) (actual time=0.398..0.398 rows=0 loops=1)
                    -> Index lookup on <materialized_subquery> using <auto_distinct_key> (PerfumeID=p.PerfumeID)  (actual time=0.398..0.398 rows=0 loops=1)
                        -> Materialize with deduplication  (cost=123..123 rows=210) (actual time=0.397..0.397 rows=0 loops=1)
                            -> Filter: ((Reviews.Feelings like '%dislike%') or (Reviews.Feelings like '%sad%'))  (cost=102 rows=210) (actual time=0.395..0.395 rows=0 loops=1)
                                -> Table scan on Reviews  (cost=102 rows=999) (actual time=0.0133..0.209 rows=1000 loops=1)
 |
```

This makes no changes to the cost of the query although we thought it would because it's a part of the JOIN, but because of the fact that it's a foreign key, the database already would index by this value, making no difference to the resulting cost.

To optimize this query, we would add an index on Review(Feelings) because it decreased the cost.

QUERY 4 INDEXING:

<u>BEFORE INDEXING:</u>

```
mysql> EXPLAIN ANALYZE SELECT p.Name AS PerfumeName, p.Brand, COUNT(r.ReviewID) AS NumberOfReviews FROM Perfumes p JOIN Reviews r ON p.PerfumeID = r.PerfumeID GROUP BY p.PerfumeID, p.Name, p.B
rand ORDER BY NumberOfReviews DESC;
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN
                                                                                                                                                                                    |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Sort: NumberOfReviews DESC  (actual time=2.58..2.59 rows=100 loops=1)
    -> Table scan on <temporary>  (actual time=2.55..2.56 rows=100 loops=1)
        -> Aggregate using temporary table  (actual time=2.55..2.55 rows=100 loops=1)
            -> Nested loop inner join  (cost=452 rows=999) (actual time=0.949..1.64 rows=1000 loops=1)
                -> Filter: (r.PerfumeID is not null)  (cost=102 rows=999) (actual time=0.937..1.16 rows=1000 loops=1)
                    -> Covering index scan on r using PerfumeID  (cost=102 rows=999) (actual time=0.936..1.11 rows=1000 loops=1)
                -> Single-row index lookup on p using PRIMARY (PerfumeID=r.PerfumeID)  (cost=0.25 rows=1) (actual time=331e-6..351e-6 rows=1 loops=1000)
 |
```

<u>After “CREATE INDEX idx_perfumes_brand ON Perfumes(Brand);”</u>

```
mysql> EXPLAIN ANALYZE SELECT p.Name AS PerfumeName, p.Brand, COUNT(r.ReviewID) AS NumberOfReviews FROM Perfumes p JOIN Reviews r ON p.PerfumeID = r.PerfumeID GROUP BY p.PerfumeID, p.Name, p.B
rand ORDER BY NumberOfReviews DESC;
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN
                                                                                                                                                                                    |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Sort: NumberOfReviews DESC  (actual time=1.68..1.69 rows=100 loops=1)
    -> Table scan on <temporary>  (actual time=1.65..1.66 rows=100 loops=1)
        -> Aggregate using temporary table  (actual time=1.65..1.65 rows=100 loops=1)
            -> Nested loop inner join  (cost=452 rows=999) (actual time=0.0376..0.741 rows=1000 loops=1)
                -> Filter: (r.PerfumeID is not null)  (cost=102 rows=999) (actual time=0.02..0.256 rows=1000 loops=1)
                    -> Covering index scan on r using PerfumeID  (cost=102 rows=999) (actual time=0.0193..0.208 rows=1000 loops=1)
                -> Single-row index lookup on p using PRIMARY (PerfumeID=r.PerfumeID)  (cost=0.25 rows=1) (actual time=344e-6..365e-6 rows=1 loops=1000)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

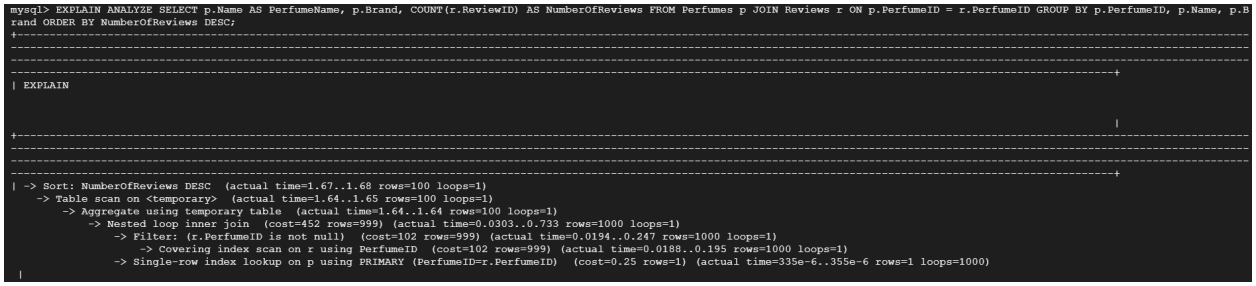
This indexing doesn’t affect the cost even though it's part of the GROUP BY. This may be because the brand isn’t as important as we initially thought, as the GROUP BY takes multiple attributes like the perfumeID and Name into account, where PerfumeId is always indexed because it’s a primary key.

<u>After “CREATE INDEX idx_perfume_name ON Perfumes(Name);”</u>

```
mysql> EXPLAIN ANALYZE SELECT p.Name AS PerfumeName, p.Brand, COUNT(r.ReviewID) AS NumberOfReviews FROM Perfumes p JOIN Reviews r ON p.PerfumeID = r.PerfumeID GROUP BY p.PerfumeID, p.Name, p.B
rand ORDER BY NumberOfReviews DESC;
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN
                                                                                                                                                                                    |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Sort: NumberOfReviews DESC  (actual time=1.66..1.67 rows=100 loops=1)
    -> Table scan on <temporary>  (actual time=1.63..1.64 rows=100 loops=1)
        -> Aggregate using temporary table  (actual time=1.63..1.63 rows=100 loops=1)
            -> Nested loop inner join  (cost=452 rows=999) (actual time=0.0297..0.726 rows=1000 loops=1)
                -> Filter: (r.PerfumeID is not null)  (cost=102 rows=999) (actual time=0.0192..0.26 rows=1000 loops=1)
                    -> Covering index scan on r using PerfumeID  (cost=102 rows=999) (actual time=0.0185..0.21 rows=1000 loops=1)
                -> Single-row index lookup on p using PRIMARY (PerfumeID=r.PerfumeID)  (cost=0.25 rows=1) (actual time=325e-6..346e-6 rows=1 loops=1000)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

This indexing doesn’t affect the cost even though it's part of the GROUP BY either. This is mainly due to the same reasoning as above, where the GROUP BY is indexed by perfumeID which is a primary key, so the database would decide to only index through that, leaving the cost unchanged.

<u>After “CREATE INDEX idx_perfume_brand_name ON Perfumes(Brand, Name);”</u>



We thought this would be valuable to try because it takes into consideration 2 components of the GROUP BY, but this also doesn't impact the cost. This must be due to the fact that since the GROUP BY has the primary key, the usefulness of indexing by these values isn't as valuable because the primary key is uniquely identifiable and can be fully relied on during the JOIN and GROUP BY.

Since all of our indexes have resulted in no change to the cost, we determined to not make any change to the original database design with no additional indexes as it seems relatively optimized.