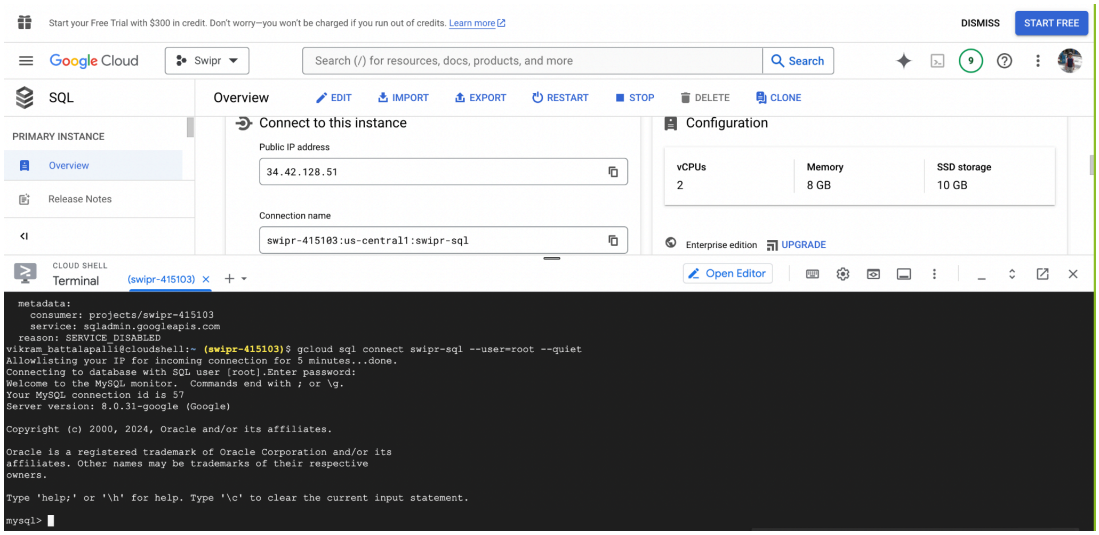
Swipr Stage 3: Database Design

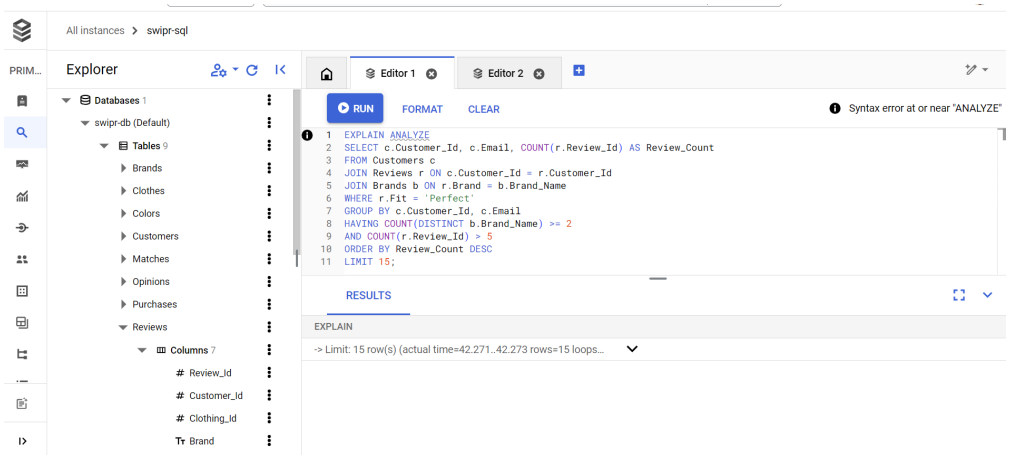
Group Members:

Raahul Rajah (rrajah2)
Ritvik Manda (rsmanda2)
Pranav Nagarajan (pranavn6)
Trivikram Battalapalli (tb17)

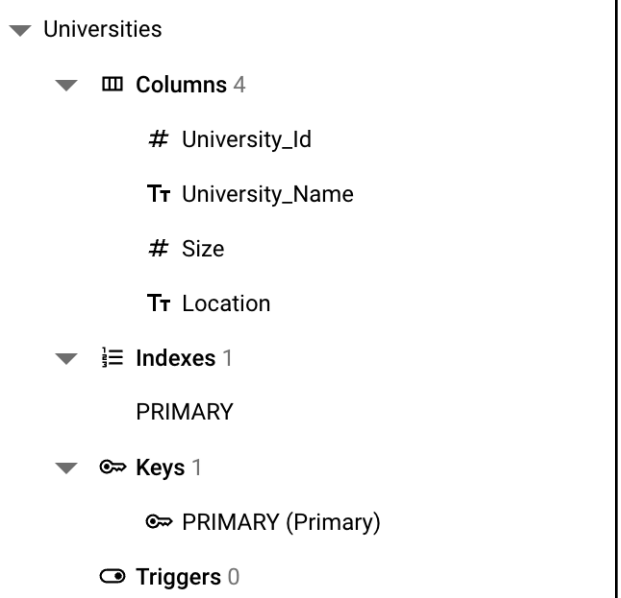
Database Implementation

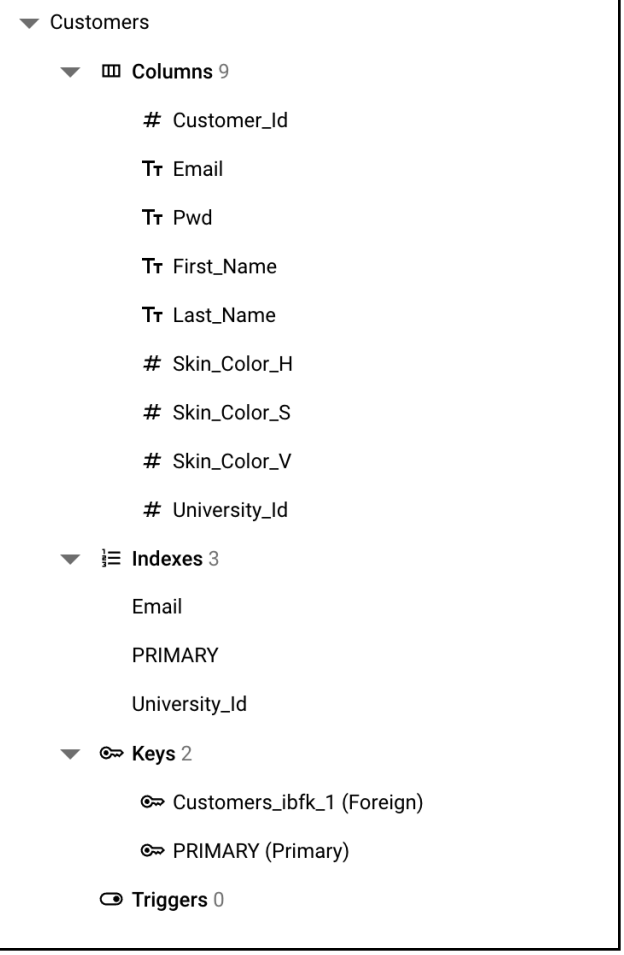
We used the inbuilt GCP Cloud SQL Studio Interface. Below is our connection proof.





DDL Commands and Row Count	Proof of Implementation

CREATE TABLE Universities (
University_Id INT PRIMARY KEY AUTO_INCREMENT,
University_Name VARCHAR(225),
Size INT,
Location VARCHAR(225)
);
`1 SELECT COUNT(University_Id) AS NUMBER`
`2 FROM Universities`
RESULTS
NUMBER
1025
	

CREATE TABLE Customers (Customer_Id INT PRIMARY KEY AUTO_INCREMENT, Email VARCHAR(225) UNIQUE, Pwd VARCHAR(225), First_Name VARCHAR(225), Last_Name VARCHAR(225), Skin_Color_H DECIMAL(5,4) CHECK (Skin_Color_H BETWEEN 0 AND 1), Skin_Color_S DECIMAL(5,4) CHECK (Skin_Color_S BETWEEN 0 AND 1), Skin_Color_V DECIMAL(5,4) CHECK (Skin_Color_V BETWEEN 0 AND 1), University_Id INT, FOREIGN KEY (University_Id) REFERENCES Universities(University_Id) ON DELETE SET NULL ON UPDATE CASCADE	

);

```
1  SELECT COUNT(Customer_Id) AS NUMBER
2  FROM Customers
```

RESULTS

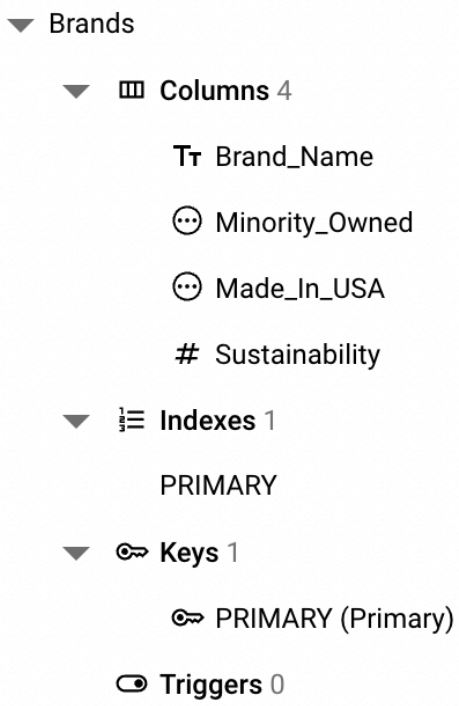
NUMBER
2499

CREATE TABLE Brands (
Brand_Name VARCHAR(100) PRIMARY KEY,
Minority_Owned BOOL NOT NULL DEFAULT FALSE,
Made_In_USA BOOL NOT NULL DEFAULT FALSE,
Sustainability INT CHECK (Sustainability BETWEEN 0 AND 100)
);

```
1  SELECT COUNT(Brand_Name) AS NUMBER
2  FROM Brands
```

RESULTS

NUMBER
4

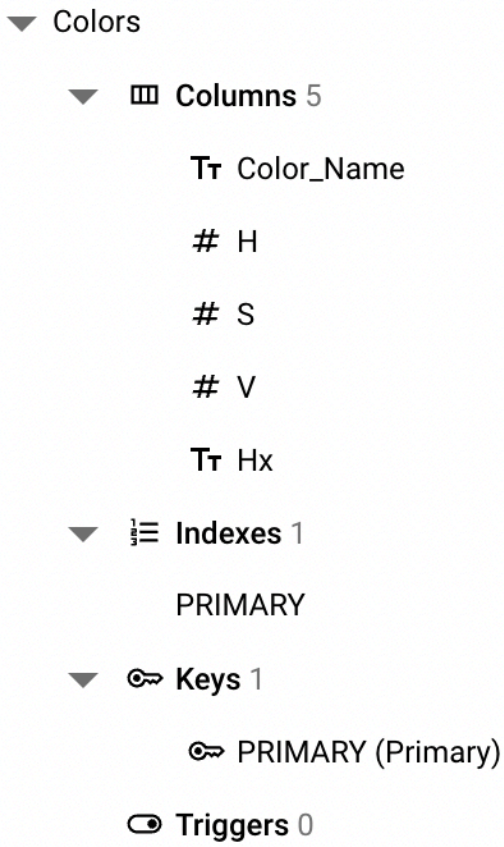


```
CREATE TABLE Colors (
  Color_Name VARCHAR(100) PRIMARY KEY,
  H DECIMAL(5,4) CHECK (H BETWEEN 0 AND 1),
  S DECIMAL(5,4) CHECK (S BETWEEN 0 AND 1),
  V DECIMAL(5,4) CHECK (V BETWEEN 0 AND 1),
  Hx VARCHAR(10)
);
```

```
1  SELECT COUNT(Color_Name) AS NUMBER
2  FROM Colors
```

RESULTS

NUMBER
865

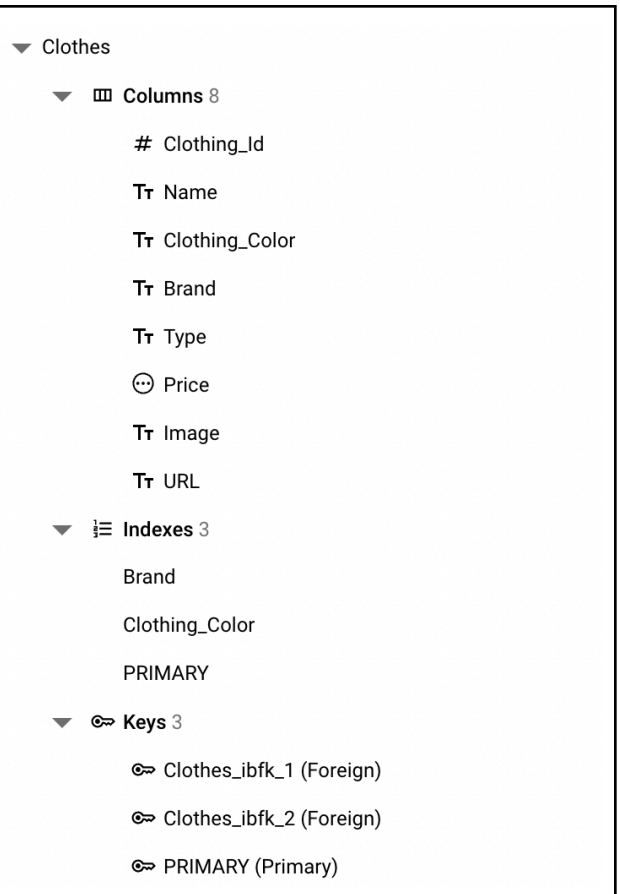


```
CREATE TABLE Clothes (
  Clothing_Id INT PRIMARY KEY AUTO_INCREMENT,
  Name VARCHAR(100),
  Clothing_Color VARCHAR(100),
  Brand VARCHAR(100),
  Type VARCHAR(20),
  Price REAL,
  Image VARCHAR(225),
  URL VARCHAR(225),
  FOREIGN KEY (Clothing_Color) REFERENCES Colors(Color_Name) ON DELETE RESTRICT,
  FOREIGN KEY (Brand) REFERENCES Brands(Brand_Name) ON DELETE RESTRICT
);
```

```
1  SELECT COUNT(Clothing_Id) AS NUMBER
2  FROM Clothes
```

RESULTS

NUMBER
9003

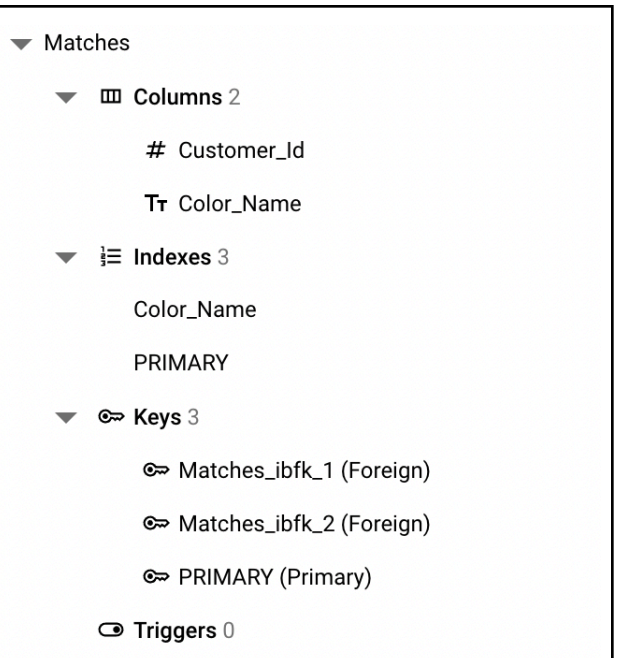


CREATE TABLE Matches (
Customer_Id INT,
Color_Name VARCHAR(100),
PRIMARY KEY (Customer_Id, Color_Name),
FOREIGN KEY (Customer_Id) REFERENCES Customers(Customer_Id) ON DELETE CASCADE,
FOREIGN KEY (Color_Name) REFERENCES Colors(Color_Name) ON DELETE RESTRICT
);

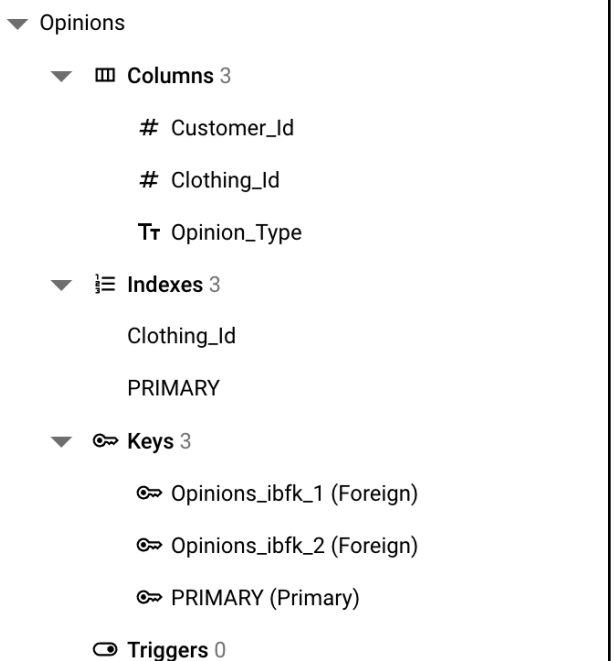
```
1  SELECT COUNT(Customer_Id) AS NUMBER
2  FROM Matches
```

RESULTS

NUMBER
49391



CREATE TABLE Opinions (
Customer_Id INT,
Clothing_Id INT,
Opinion_Type VARCHAR(225),
PRIMARY KEY (Customer_Id, Clothing_Id),
FOREIGN KEY (Customer_Id) REFERENCES Customers(Customer_Id) ON DELETE CASCADE,
FOREIGN KEY (Clothing_Id) REFERENCES Clothes(Clothing_Id) ON DELETE CASCADE



);

```
1  SELECT COUNT(Customer_Id) AS NUMBER
2  FROM Opinions
```

RESULTS

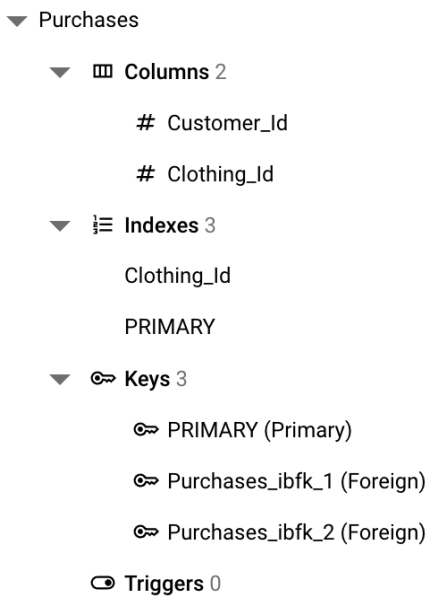
NUMBER
19972

CREATE TABLE Purchases (
Customer_Id INT,
Clothing_Id INT,
PRIMARY KEY (Customer_Id, Clothing_Id),
FOREIGN KEY (Customer_Id) REFERENCES Customers(Customer_Id),
FOREIGN KEY (Clothing_Id) REFERENCES Clothes(Clothing_Id) ON DELETE CASCADE
);

```
1  SELECT COUNT(Customer_Id) AS NUMBER
2  FROM Purchases
```

RESULTS

NUMBER
4995

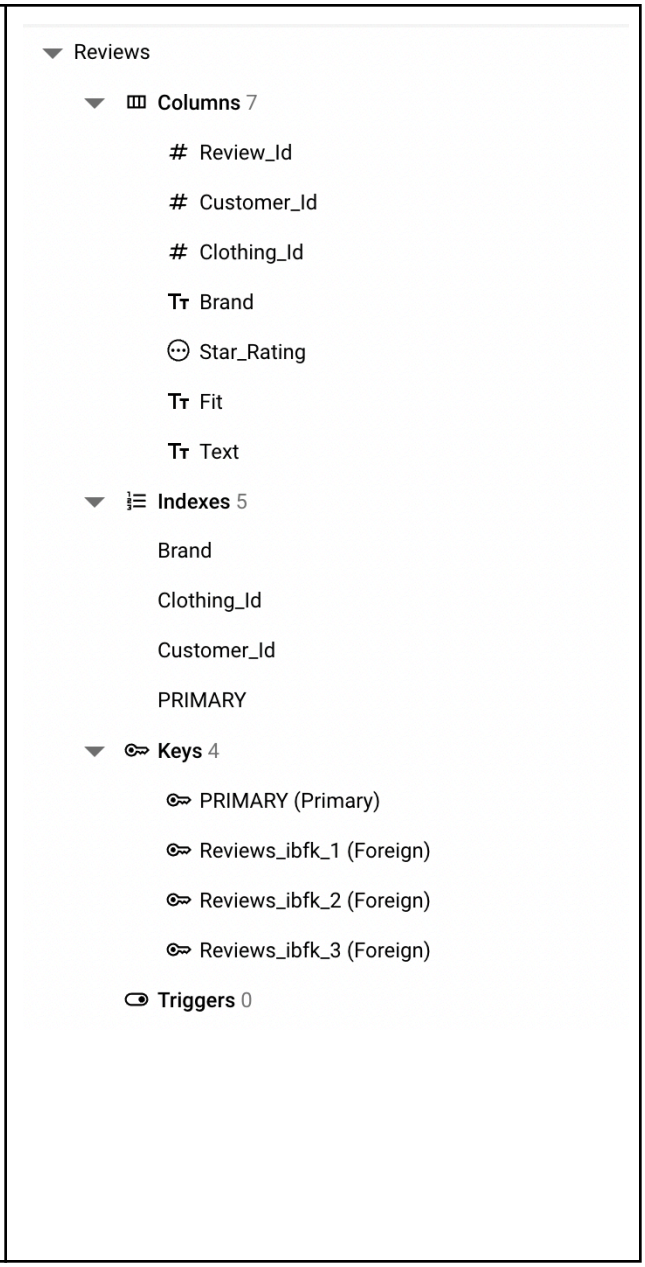


```
CREATE TABLE Reviews (
  Review_Id INT PRIMARY KEY AUTO_INCREMENT,
  Customer_Id INT,
  Clothing_Id INT,
  Brand VARCHAR(100),
  Star_Rating REAL CHECK (Star_Rating BETWEEN 1 AND 5),
  Fit VARCHAR(100),
  Text VARCHAR(1000),
  FOREIGN KEY (Customer_Id) REFERENCES Customers(Customer_Id),
  FOREIGN KEY (Clothing_Id) REFERENCES Clothes(Clothing_Id) ON DELETE CASCADE,
  FOREIGN KEY (Brand) REFERENCES Brands(Brand_Name) ON DELETE RESTRICT
);
```

```
1  SELECT COUNT(Review_Id) AS NUMBER
2  FROM Reviews
```

RESULTS

NUMBER
10159



Data Sources:

All of the data pertaining to the Clothing, Brands, and Universities is real data. The Colors table includes over 800 real colors that are used to represent the clothing. The entries in the Reviews table are real reviews of items belonging to Nike and other popular brands. For the Users table, we used an online dataset of 2500 sample users, and assigned them random skin colors from a set of 25 common skin colors. Then, we generated the matching colors in our Colors table based on our color matching algorithm to match users to clothing colors based on skin color and fill the Matches table. The Purchases and Opinions tables are randomly generated.

Advanced SQL Queries

#1 SQL Code

This query will return the clothing Id, name, price, brand, sustainability score of the brand, and the average star rating of the clothing items that cost more than $50, belong to a brand that has a sustainability score over 70, and have an average star rating over 3.5/5 stars. The results are ordered by the brand. This is useful for our app because it gives a sample scenario where the user wants to filter the clothes based on various conditions. We use JOINs and aggregation with GROUP BY in this query.

```sql
SELECT
 cl.Clothing_Id,
 cl.Name,
 cl.Price,
 cl.Brand,
 br.Sustainability,
 AVG(r.Star_Rating) AS Avg_Star_Rating
FROM Clothes as cl
JOIN Brands as br ON cl.Brand = br.Brand_Name
JOIN Reviews as r ON cl.Clothing_Id = r.Clothing_Id
WHERE
  cl.Price > 50
  AND br.Sustainability > 70
GROUP BY cl.Clothing_Id
HAVING AVG(r.Star_Rating) > 3.5
ORDER BY cl.Brand
LIMIT 15;
```

#1 SQL Output Screenshot

Clothing_Id	Name	Price	Brand	Sustainability	Avg_Star_Rating
3559	Men's HIIT Black Designed for Training HIIT Training Shorts	55	Adidas	75	3.5833333333333335
3433	Men's Cycling Black Essentials 3-Stripes Padded Cycling Bib Shorts	160	Adidas	75	3.8823529411764706
3394	Men's HIIT Black Designed for Training HIIT Workout HEAT.RDY Shorts	55	Adidas	75	3.7222222222222223
3293	Y-3 Purple Y-3 Organic Cotton Terry Crew Sweater	250	Adidas	75	3.8
3289	Men's Sportswear White Lounge Fleece Bomber Jacket With Zip Opening	100	Adidas	75	3.8
3278	Men's Soccer Black Tiro 23 League Sweat Hoodie	65	Adidas	75	3.6
3272	Men's Golf White Lightweight Half-Zip Top	70	Adidas	75	3.9411764705882355
3259	Men's Sportswear Green LA Graphic Hoodie	65	Adidas	75	3.5789473684210527
3187	Y-3 Black Y-3 Relaxed Short Sleeve Tee	100	Adidas	75	4.117647058823529
3175	Men's Golf Green adidas x Malbon Cardigan	300	Adidas	75	3.619047619047619
3111	Men's Basketball White Washington HBE Jersey	90	Adidas	75	3.8333333333333335
3109	Y-3 Black Y-3 Crepe Jersey Long Sleeve Tee	230	Adidas	75	3.764705882352941
3005	Men's Golf White Core adidas Performance Primegreen Polo Shirt	60	Adidas	75	4
3001	Men's Golf Grey Two-Color Striped Polo Shirt	60	Adidas	75	3.9444444444444446
2897	Men's Soccer Black Minnesota United FC 24/25 Home Jersey	100	Adidas	75	3.5217391304347827

#2 SQL Code

This query returns all information about clothing items along with their respective count of likes. The items are filtered based on customer opinions of all users of the app and are limited to a specific university's customers. The results are sorted by the count of likes in descending order and then by the clothing brand. The use case of this is when a customer wants to view the most popular clothing items for students who attend their university. We use JOINs, a subquery, and aggregation with GROUP BY in this query.

```
SELECT Cl.*, LikesCount
FROM Clothes AS Cl
JOIN (
  SELECT Clothing_Id, COUNT(Opinion_Type) AS LikesCount
  FROM Opinions
  WHERE Opinion_Type = 'L' OR Opinion_Type = 'S'
  GROUP BY Clothing_Id
) AS Likes ON Cl.Clothing_Id = Likes.Clothing_Id
JOIN Opinions AS Op ON Cl.Clothing_Id = Op.Clothing_Id
JOIN Customers AS Cu ON Op.Customer_Id = Cu.Customer_Id
WHERE Cu.University_Id = 3225
HAVING Likes.LikesCount > 0
ORDER BY LikesCount DESC, Cl.Brand
LIMIT 15;
```

#2 SQL Output Screenshot

Clothing_Id	Name	Clothing_Color	Brand	Type	Price	Image	
5755	Philadelph…	red_devil	Nike	Tops	40	https://static.nike.com/a/images/c_limit,w_592,f_auto/t_product_v1/…	
238	M's Terravi…	rifle_green	Patagonia	Bottoms	109	https://www.patagonia.com/dw/image/v2/BDJB_PRD/on/demandwa…	
1392	M's Pack I…	dark_tan	Patagonia	Outerwear	199	https://www.patagonia.com/dw/image/v2/BDJB_PRD/on/demandwa…	
2592	Men's Soc…	dim_gray	Adidas	Tops	55	https://assets.adidas.com/images/w_383,h_383,f_auto,q_auto,fl_loss…	
7017	Nike Tennis	red_ncs	Nike	Outerwear	65	https://static.nike.com/a/images/c_limit,w_592,f_auto/t_product_v1/…	
278	M's All Se…	davy_s_grey	Patagonia	Bottoms	79	https://www.patagonia.com/dw/image/v2/BDJB_PRD/on/demandwa…	
1323	M's Nano-…	purple_taupe	Patagonia	Outerwear	249	https://www.patagonia.com/dw/image/v2/BDJB_PRD/on/demandwa…	
1049	M's Box Q…	dark_electric_blue	Patagonia	Outerwear	249	https://www.patagonia.com/dw/image/v2/BDJB_PRD/on/demandwa…	
810	M's Capile…	copper_rose	Patagonia	Tops	59	https://www.patagonia.com/dw/image/v2/BDJB_PRD/on/demandwa…	
4191	Nike	jet	Nike	Tops	35	https://static.nike.com/a/images/c_limit,w_592,f_auto/t_product_v1/…	
6043	Nike Dri-Fl…	anti_flash_white	Nike	Tops	35	https://static.nike.com/a/images/c_limit,w_592,f_auto/t_product_v1/…	
6325	Pumas UN…	tan	Nike	Tops	35	https://static.nike.com/a/images/c_limit,w_592,f_auto/t_product_v1/…	
6900	Pittsburgh…	jet	Nike	Outerwear	85	https://static.nike.com/a/images/c_limit,w_592,f_auto/t_product_v1/…	
1853	AIRism Ful…	licorice	Uniqlo	Tops	0	https://image.uniqlo.com/UQ/ST3/WesternCommon/imagesgoods/4…	
2403	Men's Spo…	denim	Adidas	Bottoms	90	https://assets.adidas.com/images/w_383,h_383,f_auto,q_auto,fl_loss…	

URL	LikesCount
https://www.nike.com/t/philadelphia-phillies-ho…	5
https://www.patagonia.com/product/mens-terrav…	5
https://www.patagonia.com/product/mens-pack-…	5
https://www.adidas.com/us/tiro-23-league-traini…	4
https://www.nike.com/t/tennis-mens-pullover-ho…	4
https://www.patagonia.com/product/mens-all-se…	4
https://www.patagonia.com/product/mens-nano-…	4
https://www.patagonia.com/product/mens-box-q…	4
https://www.patagonia.com/product/mens-capil…	4
https://www.nike.com/t/mens-baseball-t-shirt-Jt…	3
https://www.nike.com/t/dri-fit-legend-mens-long-…	3
https://www.nike.com/t/pumas-unam-mercurial-…	3
https://www.nike.com/t/pittsburgh-pirates-authe…	3
https://www.uniqlo.com/us/en/products/E45783…	3
https://www.adidas.com/us/z.n.e.-premium-pant…	2

#3 SQL Code

This query will return all information about the clothing items that match the user according to our color matching algorithm and belong to a brand with a sustainability score >= 65. The results are ordered by the price. We utilized customer 7 as an example, showcasing the clothing items that match them based on our color matching algorithm. This is useful for our app because it gives a scenario where the user sees their recommended clothing items based on their skin tone. We use JOINs and a subquery in this query.

```
SELECT cl.*, c.Hx
FROM Clothes as cl
```

```sql
JOIN Colors as c ON cl.Clothing_Color = c.Color_Name
JOIN Brands as b ON cl.Brand = b.Brand_Name
WHERE c.Color_Name IN (
   SELECT Color_Name
   FROM Matches
   WHERE Customer_Id = 7
)
AND b.Sustainability >= 65
ORDER BY cl.Price
LIMIT 15;
```

#3 SQL Output Screenshot

Clothing_Id	Name	Clothing_Color	Brand	Type	Price	Image
2230	Oxford Str…	languid_lavender	Uniqlo	Tops	0	https://image.uniqlo.com/UQ/ST3/us/imagesgoods/462369/item/us…
1835	Sweatpan…	languid_lavender	Uniqlo	Tops	0	https://image.uniqlo.com/UQ/ST3/us/imagesgoods/466771/item/us…
2124	Oxford Str…	languid_lavender	Uniqlo	Tops	0	https://image.uniqlo.com/UQ/ST3/us/imagesgoods/462369/item/us…
2229	Premium …	languid_lavender	Uniqlo	Tops	0	https://image.uniqlo.com/UQ/ST3/us/imagesgoods/455957/item/us…
2039	DRY-EX Cr…	viridian	Uniqlo	Tops	0	https://image.uniqlo.com/UQ/ST3/us/imagesgoods/456772/item/us…
1916	AIRism C…	pale_pink	Uniqlo	Tops	0	https://image.uniqlo.com/UQ/ST3/us/imagesgoods/465196/item/us…
2221	Supima® …	languid_lavender	Uniqlo	Tops	0	https://image.uniqlo.com/UQ/ST3/us/imagesgoods/455365/item/us…
5375	Nike Spor…	languid_lavender	Nike	Tops	30	https://static.nike.com/a/images/c_limit,w_592,f_auto/t_product_v1/…
4616	Nike Dri-FIT	pale_pink	Nike	Tops	35	https://static.nike.com/a/images/c_limit,w_592,f_auto/t_product_v1/…
6029	Nike Spor…	pale_pink	Nike	Tops	35	https://static.nike.com/a/images/c_limit,w_592,f_auto/t_product_v1/…
6055	Nike Spor…	pale_pink	Nike	Tops	35	https://static.nike.com/a/images/c_limit,w_592,f_auto/t_product_v1/…
5414	Nike Spor…	languid_lavender	Nike	Tops	35	https://static.nike.com/a/images/c_limit,w_592,f_auto/t_product_v1/…
4834	Nike Spor…	viridian	Nike	Tops	35	https://static.nike.com/a/images/c_limit,w_592,f_auto/t_product_v1/…
5396	Sabrina	languid_lavender	Nike	Tops	38	https://static.nike.com/a/images/c_limit,w_592,f_auto/t_product_v1/…
4811	Nike Spor…	viridian	Nike	Tops	40	https://static.nike.com/a/images/c_limit,w_592,f_auto/t_product_v1/…

URL	Hx
https://www.uniqlo.com/us/en/products/E46236...	#d6cadd
https://www.uniqlo.com/us/en/products/E46677...	#d6cadd
https://www.uniqlo.com/us/en/products/E46236...	#d6cadd
https://www.uniqlo.com/us/en/products/E45595...	#d6cadd
https://www.uniqlo.com/us/en/products/E45677...	#40826d
https://www.uniqlo.com/us/en/products/E46519...	#fadadd
https://www.uniqlo.com/us/en/products/E45536...	#d6cadd
https://www.nike.com/t/sportswear-swoosh-men...	#d6cadd
https://www.nike.com/t/dri-fit-mens-basketball-t-...	#fadadd
https://www.nike.com/t/sportswear-mens-tank-5...	#fadadd
https://www.nike.com/t/sportswear-mens-tank-5...	#fadadd
https://www.nike.com/t/sportswear-mens-tank-5...	#d6cadd
https://www.nike.com/t/sportswear-club-mens-lo...	#40826d
https://www.nike.com/t/sabrina-womens-boxy-b...	#d6cadd
https://www.nike.com/t/sportswear-mens-t-shirt-...	#40826d

#4 SQL Code

This query will return the customer Id, email, first name, last name, and review count of customers who have written at least 5 reviews and have reviewed at least 2 brands of clothing. We exclusively consider reviews that rate the item's fit as perfect. This approach is taken because lower ratings might stem from purchasers selecting incorrect sizes, rather than the item itself being of poor quality. This is useful for our app as customers can view top reviewers in the app to get styling advice from experienced stylists and customers. We use JOINs and aggregation with GROUP BY in this query.

```
SELECT c.Customer_Id, c.Email, c.First_Name, c.Last_Name, COUNT(r.Review_Id) AS
Review_Count
FROM Customers c
JOIN Reviews r ON c.Customer_Id = r.Customer_Id
JOIN Brands b ON r.Brand = b.Brand_Name
WHERE r.Fit = 'Perfect'
GROUP BY c.Customer_Id, c.Email
HAVING COUNT(DISTINCT b.Brand_Name) >= 2
AND COUNT(r.Review_Id) > 5
ORDER BY Review_Count DESC
LIMIT 15;
```

#4 SQL Output Screenshot

Customer_Id	Email	First_Name	Last_Name	Review_Count
1452	cristal@cox.net	Cristal	Samara	8
1044	erick.ferencz@aol.com	Erick	Ferencz	8
662	gerixon@gmail.com	Gilberto	Erixon	8
320	oramerez@yahoo.com	Oliva	Ramerez	8
2481	lmckenzie@example.org	James	Atkins	7
2401	mendezbrenda@example.com	Alyssa	Robles	7
2025	reynoldsstacey@example.net	Ariel	Wilkinson	7
1948	ernestblake@example.net	Lynn	Cordova	7
1798	margaretlane@example.org	Stacy	Roach	7
1753	diane97@example.org	Jamie	Logan	7
1624	nicolas31@example.com	Tracie	Mooney	7
497	leslee_matsuno@matsuno.org	Leslee	Matsuno	7
1545	diana26@example.net	Olivia	Duke	7
1494	lai@gmail.com	Lai	Harabedian	7
1489	lawrence.lorens@hotmail.com	Lawrence	Lorens	7

Indexing *

Indexing for Query #1

By analyzing this query, we can see that the only attributes that can be indexed are Star_Rating, Price, and Sustainability, as every other attribute is either a primary key or foreign key. We first indexed the attribute Star_Rating, but after comparing the results, we saw that the before and after costs did not change at all. This is most probably since we are performing an "average" for all of the Star_Rating elements, the program is not performing a lookup function, but rather just getting all of the values needed to compute the average. Hence, the ordering doesn't matter and therefore an index is not needed. As a result, we dropped the index and kept the original schema of the database. See the before and after below:

Before	Nested loop inner join (cost=3762.31 rows=542) (actual time=0.140..32.760 rows=2254 loops=1)
After	Nested loop inner join (cost=3762.31 rows=542) (actual time=0.093..34.885 rows=2254 loops=1)

Since indexing Star_Rating didn't result in a gain in performance, we do not need to include it for any of the permutations of the attributes. Next, we checked to see if indexing the Price would change the performance - it did not. See the before and after below. We can see that the cost for both before and after are the exact same, which suggests that indexing Price did not have any impact. The reasons for this are Price's uniform distribution and its low selectivity. As a result, we dropped the index and kept the original schema of the database.

Before	Nested loop inner join (cost=3762.31 rows=542) (actual time=0.140..32.760 rows=2254 loops=1)
After	Nested loop inner join (cost=3762.31 rows=1040) (actual time=0.102..46.207 rows=2254 loops=1)

For the same reason why we didn't include Star_Rating in the permutations, we are also not including Price. Next, we tried to index Sustainability. See the before and after below. As you can see, by indexing Sustainability, the cost spiked up. This is because the sustainability attribute only applies to four brands, which itself is a very small table. This renders the indexing useless since the program can just scan the table as a whole, which is only four entries. As a result, we dropped the index on Sustainability and kept the original schema of the database.

Before	Nested loop inner join (cost=3762.31 rows=542) (actual time=0.140..32.760 rows=2254 loops=1)
After	Nested loop inner join (cost=5549.14 rows=2437) (actual time=0.114..27.086 rows=2254 loops=1)

Indexing for Query #2

By analyzing this query, we can see that almost every attribute cannot be indexed since they are either primary keys or foreign keys, such as Clothing_Id, Customer_Id, University_Id, etc. However, the only attribute that could be indexed was Opinion_Type from Opinions. After creating an index on Opinion_Type, we saw that the total cost of the query increased by a factor of over 4. Please review the before and after results.

Before	Stream results (cost=12680.41 rows=125728) (actual time=40.495..40.811 rows=35 loops=1) Group aggregate: count(Opinions.Opinion_Type) (cost=2229.56 rows=3521)
After	Stream results (cost=51118.85 rows=510112) (actual time=40.688..40.974 rows=35 loops=1) Group aggregate: count(Opinions.Opinion_Type) (cost=3306.05 rows=14286)

We can see that 51118.85 > 12680.41, both of which represent the total cost of the query, which caused us to drop the index and keep the original schema of the database. Additionally, we can see that the group aggregate cost after indexing causes a spike in cost as well, while also increasing the total number of rows the query parses through. This change is mostly due to the fact that Opinion_Type only has three options, L for Likes, D for Dislikes, and S for Superlikes. A piece of clothing can be attributed to only one of these three options and as a result, there is really no need to order these choices, rendering the indexing useless and a waste of resources.

In addition to Opinion_Type, we also have other attributes we can index such as Price and Clothing_Type, referred to as Type in the "Clothes" table. Let's start with Price. We checked to see if indexing the Price would change the performance - it did not. See the before and after below. We can see that the cost for both before and after are the exact same, which suggests that indexing Price did not have any impact. As a result, we dropped the index and kept the original schema of the database.

Before	Stream results (cost=12680.41 rows=125728) (actual time=40.495..40.811 rows=35 loops=1)
After	Stream results (cost=12680.01 rows=125728) (actual time=69.944..70.388 rows=35 loops=1)

Now, we checked to see if Clothing_Type had any impact on performance. See the table below. We can see that the performance barely changed. This change is mostly due to the fact that Type only has three options, Tops, Bottoms, and Outerwear. A piece of clothing can be attributed to only one of these three options, and as a result, there is really no need to order these choices, rendering the indexing useless and a waste of resources.

Before	Stream results (cost=12680.41 rows=125728) (actual time=40.495..40.811 rows=35 loops=1)
After	Stream results (cost=12680.01 rows=125728) (actual time=45.631..45.894 rows=35 loops=1)

Indexing for Query #3

By analyzing this query, we can determine the only elements that can be indexed: Price and Sustainability. Let's start with Price. Below shows the before and after indexing Price. We can see that the cost for both before and after are the exact same, which suggests that indexing Price did not have any impact. As a result, we dropped the index and kept the original schema of the database.

Before	Stream results (cost=106.02 rows=70) (actual time=0.111..1.377 rows=55 loops=1)
After	Stream results (cost=106.02 rows=70) (actual time=0.074..0.816 rows=55 loops=1)

Since Price had no impact on cost, we don't need to include it in our permutations of the attributes to be indexed. Below represents the table for before and after indexing Sustainability. As you can see, by indexing Sustainability, the cost spiked up. This is because the sustainability attribute only applies to four brands, which itself is a very small table. This renders the indexing useless since the program can just scan the table as a whole, which is only four entries. As a result, we dropped the index on Sustainability and kept the original

schema of the database.

Before	Stream results (cost=106.02 rows=70) (actual time=0.111..1.377 rows=55 loops=1)
After	Stream results (cost=153.18 rows=210) (actual time=0.073..0.472 rows=55 loops=1)

Now, we checked to see if Clothing_Type had any impact on performance. See the table below. We can see that the performance barely changed. This change is mostly due to the fact that Type only has three options, Tops, Bottoms, Outerwear. A piece of clothing can be attributed to only one of these three options and as a result, there is really no need to order these choices. However, the slight change can be attributed to the fact that since the Clothes table itself is very large, the ordering of the type can be slightly helpful. Thus, rendering the indexing is slightly useful - as a result, we decided to keep this index in our schema.

Before	Stream results (cost=106.02 rows=70) (actual time=0.111..1.377 rows=55 loops=1)
After	Stream results (cost=104.39 rows=70) (actual time=0.077..0.810 rows=55 loops=1)

Indexing for Query #4

Analyzing this query, we can determine that almost every attribute used is either a primary key or foreign key except the Fit attribute in Reviews. The before and after results of indexing this attribute are shown below. As you can see, by indexing Fit, the cost spiked up. This is because the Fit attribute only possesses three unique values, which are Perfect, Large, and Small. A review can be attributed to only one of these three options and as a result, there is really no need to order these choices, rendering the indexing useless and a waste of resources. As a result, we dropped the index on Fit and kept the original schema of the database.

Before	Stream results (cost=1607.90 rows=975) (actual time=0.277..29.024 rows=5534 loops=1)
After	Stream results (cost=3203.55 rows=5534) (actual time=0.281..29.320 rows=5534 loops=1)

Next, we decided to index both customer First_Name and customer Last_Name. Let's start with First_Name. See the results below. We can see that indexing First_Name does not change the performance of the query. This is most probably due to the fact that every Customer_Id attributes to a First_Name, and Customer_Id is already an index itself because it is a primary key. Thus, the index for First_Name is useless since the ordering is already being done more accurately by Customer_Id. As a result, we dropped this index and kept our original database schema.

Before	Stream results (cost=1607.90 rows=975) (actual time=0.277..29.024 rows=5534 loops=1)

After	Stream results (cost=1607.90 rows=975) (actual time=0.364..29.939 rows=5534 loops=1)

Next, let's look at the results of indexing the Last_Name below. We can see that indexing Last_Name does not change the performance of the query. This is most probably due to the fact that every Customer_Id attributes to a Last_Name, and Customer_Id is already an index itself because it is a primary key. Thus, the index for Last_Name is useless since the ordering is already being done more accurately by Customer_Id. As a result, we dropped this index and kept our original database schema.

Before	Stream results (cost=1607.90 rows=975) (actual time=0.277..29.024 rows=5534 loops=1)
After	Stream results (cost=1607.90 rows=975) (actual time=0.230..29.130 rows=5534 loops=1)

* Full EXPLAIN ANALYZE Results

Indexing for Query #1

Original

-> Limit: 15 row(s) (actual time=34.693..34.695 rows=15 loops=1) -> Sort: cl.Brand (actual time=34.692..34.694 rows=15 loops=1) -> Filter: (avg(r.Star_Rating) > 3.5) (actual time=34.581..34.636 rows=62 loops=1) -> Table scan on <temporary> (actual time=34.573..34.618 rows=125 loops=1) -> Aggregate using temporary table (actual time=34.567..34.567 rows=125 loops=1) -> Nested loop inner join (cost=3762.31 rows=542) (actual time=0.140..32.760 rows=2254 loops=1) -> Inner hash join (no condition) (cost=674.87 rows=6499) (actual time=0.123..17.043 rows=30477 loops=1) -> Index range scan on r using Clothing_Id over (NULL < Clothing_Id), with index condition: (r.Clothing_Id is not null) (cost=191.94 rows=4875) (actual time=0.025..14.068 rows=10159 loops=1) -> Hash -> Filter: (br.Sustainability > 70) (cost=0.65 rows=1) (actual time=0.063..0.069 rows=3 loops=1) -> Table scan on br (cost=0.65 rows=4) (actual time=0.061..0.065 rows=4 loops=1) -> Filter: ((cl.Brand = br.Brand_Name) and (cl.Price > 50)) (cost=0.19 rows=0.08) (actual time=0.000..0.000 rows=0 loops=30477) -> Single-row index lookup on cl using PRIMARY (Clothing_Id=r.Clothing_Id) (cost=0.19 rows=1) (actual time=0.000..0.000 rows=1 loops=30477)

After Star_Rating Indexing

-> Limit: 15 row(s) (actual time=36.993..36.996 rows=15 loops=1) -> Sort: cl.Brand (actual time=36.992..36.994 rows=15 loops=1) -> Filter: (avg(r.Star_Rating) > 3.5) (actual time=36.769..36.935 rows=62 loops=1) -> Table scan on <temporary> (actual time=36.761..36.901 rows=125 loops=1) -> Aggregate using temporary table (actual time=36.755..36.755 rows=125 loops=1) -> Nested loop inner join (cost=3762.31 rows=542) (actual time=0.093..34.885 rows=2254 loops=1) -> Inner hash join (no condition) (cost=674.87 rows=6499) (actual time=0.068..18.561 rows=30477 loops=1) -> Index range scan on r using Clothing_Id over (NULL < Clothing_Id), with index condition: (r.Clothing_Id is not null) (cost=191.94 rows=4875) (actual time=0.024..15.657 rows=10159 loops=1) -> Hash -> Filter: (br.Sustainability > 70) (cost=0.65 rows=1) (actual time=0.029..0.033 rows=3 loops=1) -> Table scan on br (cost=0.65 rows=4) (actual time=0.028..0.031 rows=4 loops=1) -> Filter: ((cl.Brand = br.Brand_Name) and (cl.Price > 50)) (cost=0.19 rows=0.08) (actual time=0.000..0.000 rows=0 loops=30477) -> Single-row index lookup on cl using PRIMARY (Clothing_Id=r.Clothing_Id) (cost=0.19 rows=1) (actual time=0.000..0.000 rows=1 loops=30477)

After Price Indexing

-> Limit: 15 row(s) (actual time=49.804..49.807 rows=15 loops=1) -> Sort: cl.Brand (actual time=49.803..49.805 rows=15 loops=1) -> Filter: (avg(r.Star_Rating) > 3.5) (actual time=49.681..49.745 rows=62 loops=1) -> Table scan on <temporary> (actual time=49.660..49.713 rows=125 loops=1) -> Aggregate using temporary table (actual time=49.654..49.654 rows=125 loops=1) -> Nested loop inner join (cost=3762.31 rows=1040) (actual time=0.102..46.207 rows=2254 loops=1) -> Inner hash join (no condition) (cost=674.87 rows=6499) (actual time=0.079..24.909 rows=30477 loops=1) -> Index range scan on r using Clothing_Id over (NULL < Clothing_Id), with index condition: (r.Clothing_Id is not null) (cost=191.94 rows=4875) (actual time=0.046..20.833 rows=10159 loops=1) -> Hash -> Filter: (br.Sustainability > 70) (cost=0.65 rows=1) (actual time=0.018..0.022 rows=3 loops=1) -> Table scan on br (cost=0.65 rows=4) (actual time=0.017..0.019 rows=4 loops=1) -> Filter: ((cl.Brand = br.Brand_Name) and (cl.Price > 50)) (cost=0.19 rows=0.2) (actual time=0.001..0.001 rows=0 loops=30477) -> Single-row index lookup on cl using PRIMARY (Clothing_Id=r.Clothing_Id) (cost=0.19 rows=1) (actual time=0.000..0.000 rows=1 loops=30477)

After Sustainability Indexing

-> Limit: 15 row(s) (actual time=29.320..29.323 rows=15 loops=1) -> Sort: cl.Brand (actual time=29.319..29.321 rows=15 loops=1) -> Filter: (avg(r.Star_Rating) > 3.5) (actual time=29.193..29.230 rows=62 loops=1) -> Table scan on <temporary> (actual time=29.186..29.213 rows=125 loops=1) -> Aggregate using temporary table (actual time=29.182..29.182 rows=125 loops=1) -> Nested loop inner join (cost=5549.14 rows=2437) (actual time=0.114..27.086 rows=2254 loops=1) -> Nested loop inner join (cost=4411.75 rows=3250) (actual time=0.105..20.411 rows=6617 loops=1) -> Filter: (r.Clothing_Id is not null) (cost=999.25 rows=9750) (actual time=0.091..4.160 rows=10159 loops=1) -> Table scan on r (cost=999.25 rows=9750) (actual time=0.090..3.344 rows=10159 loops=1) -> Filter: ((cl.Price > 50) and (cl.Brand is not null)) (cost=0.25 rows=0.3) (actual time=0.001..0.001 rows=1 loops=10159) -> Single-row index lookup on cl using PRIMARY (Clothing_Id=r.Clothing_Id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=10159) ->

Filter: (br.Sustainability > 70) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=0 loops=6617) -> Single-row index lookup on br using PRIMARY (Brand_Name=cl.Brand) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=6617)

Indexing for Query #2

Original

-> Limit: 15 row(s) (actual time=40.863..40.867 rows=15 loops=1) -> Sort: Likes.LikesCount DESC, Cl.Brand, limit input to 15 row(s) per chunk (actual time=40.862..40.865 rows=15 loops=1) -> Filter: (Likes.LikesCount > 0) (actual time=40.498..40.819 rows=35 loops=1) -> Stream results (cost=12680.41 rows=125728) (actual time=40.495..40.811 rows=35 loops=1) -> Nested loop inner join (cost=12680.41 rows=125728) (actual time=40.487..40.761 rows=35 loops=1) -> Nested loop inner join (cost=18.35 rows=36) (actual time=0.058..0.231 rows=36 loops=1) -> Nested loop inner join (cost=5.45 rows=36) (actual time=0.045..0.077 rows=36 loops=1) -> Covering index lookup on Cu using University_Id (University_Id=3225) (cost=1.11 rows=3) (actual time=0.026..0.031 rows=3 loops=1) -> Covering index lookup on Op using PRIMARY (Customer_Id=Cu.Customer_Id) (cost=0.65 rows=12) (actual time=0.011..0.014 rows=12 loops=3) -> Single-row index lookup on Cl using PRIMARY (Clothing_Id=Op.Clothing_Id) (cost=0.26 rows=1) (actual time=0.004..0.004 rows=1 loops=36) -> Index lookup on Likes using <auto_key0> (Clothing_Id=Op.Clothing_Id) (actual time=1.125..1.126 rows=1 loops=36) -> Materialize (cost=2581.67..2581.67 rows=3521) (actual time=40.416..40.416 rows=7177 loops=1) -> Group aggregate: count(Opinions.Opinion_Type) (cost=2229.56 rows=3521) (actual time=0.329..33.709 rows=7177 loops=1) -> Filter: ((Opinions.Opinion_Type = 'L') or (Opinions.Opinion_Type = 'S')) (cost=1877.45 rows=3521) (actual time=0.322..31.667 rows=14286 loops=1) -> Index scan on Opinions using Clothing_Id (cost=1877.45 rows=18532) (actual time=0.319..28.498 rows=19972 loops=1)

After Opinion_Type Indexing

-> Limit: 15 row(s) (actual time=41.059..41.063 rows=15 loops=1) -> Sort: Likes.LikesCount DESC, Cl.Brand, limit input to 15 row(s) per chunk (actual time=41.059..41.062 rows=15 loops=1) -> Filter: (Likes.LikesCount > 0) (actual time=40.691..40.982 rows=35 loops=1) -> Stream results (cost=51118.85 rows=510112) (actual time=40.688..40.974 rows=35 loops=1) -> Nested loop inner join (cost=51118.85 rows=510112) (actual time=40.679..40.926 rows=35 loops=1) -> Nested loop inner join (cost=18.35 rows=36) (actual time=0.044..0.199 rows=36 loops=1) -> Nested loop inner join (cost=5.45 rows=36) (actual time=0.032..0.065 rows=36 loops=1) -> Covering index lookup on Cu using University_Id (University_Id=3225) (cost=1.11 rows=3) (actual time=0.018..0.023 rows=3 loops=1) -> Covering index lookup on Op using PRIMARY (Customer_Id=Cu.Customer_Id) (cost=0.65 rows=12) (actual time=0.009..0.013 rows=12 loops=3) -> Single-row index lookup on Cl using PRIMARY (Clothing_Id=Op.Clothing_Id) (cost=0.26 rows=1) (actual time=0.003..0.003 rows=1 loops=36) -> Index lookup on Likes using <auto_key0> (Clothing_Id=Op.Clothing_Id) (actual time=1.131..1.131 rows=1 loops=36) -> Materialize (cost=4734.65..4734.65 rows=14286) (actual

time=40.621..40.621 rows=7177 loops=1) -> Group aggregate: count(Opinions.Opinion_Type) (cost=3306.05 rows=14286) (actual time=0.280..33.805 rows=7177 loops=1) -> Filter: ((Opinions.Opinion_Type = 'L') or (Opinions.Opinion_Type = 'S')) (cost=1877.45 rows=14286) (actual time=0.274..31.811 rows=14286 loops=1) -> Index scan on Opinions using Clothing_Id (cost=1877.45 rows=18532) (actual time=0.269..28.431 rows=19972 loops=1)

After Price Indexing

-> Limit: 15 row(s) (actual time=70.451..70.458 rows=15 loops=1) -> Sort: Likes.LikesCount DESC, Cl.Brand, limit input to 15 row(s) per chunk (actual time=70.451..70.456 rows=15 loops=1) -> Filter: (Likes.LikesCount > 0) (actual time=69.947..70.400 rows=35 loops=1) -> Stream results (cost=12680.01 rows=125728) (actual time=69.944..70.388 rows=35 loops=1) -> Nested loop inner join (cost=12680.01 rows=125728) (actual time=69.933..70.316 rows=35 loops=1) -> Nested loop inner join (cost=17.95 rows=36) (actual time=0.086..0.330 rows=36 loops=1) -> Nested loop inner join (cost=5.45 rows=36) (actual time=0.072..0.147 rows=36 loops=1) -> Covering index lookup on Cu using University_Id (University_Id=3225) (cost=1.11 rows=3) (actual time=0.015..0.048 rows=3 loops=1) -> Covering index lookup on Op using PRIMARY (Customer_Id=Cu.Customer_Id) (cost=0.65 rows=12) (actual time=0.026..0.031 rows=12 loops=3) -> Single-row index lookup on Cl using PRIMARY (Clothing_Id=Op.Clothing_Id) (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=36) -> Index lookup on Likes using <auto_key0> (Clothing_Id=Op.Clothing_Id) (actual time=1.943..1.944 rows=1 loops=36) -> Materialize (cost=2581.67..2581.67 rows=3521) (actual time=69.834..69.834 rows=7177 loops=1) -> Group aggregate: count(Opinions.Opinion_Type) (cost=2229.56 rows=3521) (actual time=0.419..57.191 rows=7177 loops=1) -> Filter: ((Opinions.Opinion_Type = 'L') or (Opinions.Opinion_Type = 'S')) (cost=1877.45 rows=3521) (actual time=0.411..53.689 rows=14286 loops=1) -> Index scan on Opinions using Clothing_Id (cost=1877.45 rows=18532) (actual time=0.405..46.840 rows=19972 loops=1)

After Type Indexing

-> Limit: 15 row(s) (actual time=46.037..46.041 rows=15 loops=1) -> Sort: Likes.LikesCount DESC, Cl.Brand, limit input to 15 row(s) per chunk (actual time=46.036..46.040 rows=15 loops=1) -> Filter: (Likes.LikesCount > 0) (actual time=45.634..45.903 rows=35 loops=1) -> Stream results (cost=12680.01 rows=125728) (actual time=45.631..45.894 rows=35 loops=1) -> Nested loop inner join (cost=12680.01 rows=125728) (actual time=45.622..45.845 rows=35 loops=1) -> Nested loop inner join (cost=17.95 rows=36) (actual time=0.060..0.195 rows=36 loops=1) -> Nested loop inner join (cost=5.45 rows=36) (actual time=0.047..0.078 rows=36 loops=1) -> Covering index lookup on Cu using University_Id (University_Id=3225) (cost=1.11 rows=3) (actual time=0.026..0.030 rows=3 loops=1) -> Covering index lookup on Op using PRIMARY (Customer_Id=Cu.Customer_Id) (cost=0.65 rows=12) (actual time=0.011..0.015 rows=12 loops=3) -> Single-row index lookup on Cl using PRIMARY (Clothing_Id=Op.Clothing_Id) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=36) -> Index lookup on Likes using <auto_key0> (Clothing_Id=Op.Clothing_Id) (actual time=1.267..1.268 rows=1 loops=36) -> Materialize (cost=2581.67..2581.67 rows=3521) (actual time=45.548..45.548 rows=7177 loops=1) -> Group aggregate: count(Opinions.Opinion_Type) (cost=2229.56 rows=3521) (actual time=0.368..37.686 rows=7177 loops=1) -> Filter: ((Opinions.Opinion_Type = 'L') or (Opinions.Opinion_Type = 'S')) (cost=1877.45 rows=3521) (actual

time=0.361..35.377 rows=14286 loops=1) -> Index scan on Opinions using Clothing_Id (cost=1877.45 rows=18532) (actual time=0.357..31.834 rows=19972 loops=1)

Indexing for Query #3

Original

-> Limit: 15 row(s) (actual time=1.426..1.432 rows=15 loops=1) -> Sort: cl.Price, limit input to 15 row(s) per chunk (actual time=1.426..1.429 rows=15 loops=1) -> Stream results (cost=106.02 rows=70) (actual time=0.111..1.377 rows=55 loops=1) -> Nested loop inner join (cost=106.02 rows=70) (actual time=0.106..1.251 rows=55 loops=1) -> Nested loop inner join (cost=8.00 rows=13) (actual time=0.063..0.219 rows=40 loops=1) -> Nested loop inner join (cost=4.16 rows=13) (actual time=0.047..0.096 rows=40 loops=1) -> Filter: (b.Sustainability >= 65) (cost=0.65 rows=1) (actual time=0.031..0.038 rows=4 loops=1) -> Table scan on b (cost=0.65 rows=4) (actual time=0.029..0.035 rows=4 loops=1) -> Covering index lookup on Matches using PRIMARY (Customer_Id=7) (cost=2.38 rows=10) (actual time=0.009..0.013 rows=10 loops=4) -> Single-row index lookup on c using PRIMARY (Color_Name=Matches.Color_Name) (cost=0.20 rows=1) (actual time=0.003..0.003 rows=1 loops=40) -> Filter: (cl.Brand = b.Brand_Name) (cost=5.29 rows=5) (actual time=0.022..0.025 rows=1 loops=40) -> Index lookup on cl using Clothing_Color (Clothing_Color=Matches.Color_Name) (cost=5.29 rows=21) (actual time=0.020..0.024 rows=6 loops=40)

After Price Indexing

-> Limit: 15 row(s) (actual time=0.849..0.853 rows=15 loops=1) -> Sort: cl.Price, limit input to 15 row(s) per chunk (actual time=0.848..0.851 rows=15 loops=1) -> Stream results (cost=106.02 rows=70) (actual time=0.074..0.816 rows=55 loops=1) -> Nested loop inner join (cost=106.02 rows=70) (actual time=0.071..0.752 rows=55 loops=1) -> Nested loop inner join (cost=8.00 rows=13) (actual time=0.049..0.139 rows=40 loops=1) -> Nested loop inner join (cost=4.16 rows=13) (actual time=0.036..0.064 rows=40 loops=1) -> Filter: (b.Sustainability >= 65) (cost=0.65 rows=1) (actual time=0.025..0.030 rows=4 loops=1) -> Table scan on b (cost=0.65 rows=4) (actual time=0.024..0.027 rows=4 loops=1) -> Covering index lookup on Matches using PRIMARY (Customer_Id=7) (cost=2.38 rows=10) (actual time=0.005..0.007 rows=10 loops=4) -> Single-row index lookup on c using PRIMARY (Color_Name=Matches.Color_Name) (cost=0.20 rows=1) (actual time=0.002..0.002 rows=1 loops=40) -> Filter: (cl.Brand = b.Brand_Name) (cost=5.29 rows=5) (actual time=0.013..0.015 rows=1 loops=40) -> Index lookup on cl using Clothing_Color (Clothing_Color=Matches.Color_Name) (cost=5.29 rows=21) (actual time=0.012..0.014 rows=6 loops=40)

After Sustainability Indexing

-> Limit: 15 row(s) (actual time=0.520..0.524 rows=15 loops=1) -> Sort: cl.Price, limit input to 15 row(s) per chunk (actual time=0.520..0.522 rows=15 loops=1) -> Stream results (cost=153.18 rows=210) (actual time=0.073..0.472 rows=55 loops=1) -> Nested loop inner join (cost=153.18

rows=210) (actual time=0.069..0.413 rows=55 loops=1) -> Nested loop inner join (cost=79.66 rows=210) (actual time=0.055..0.314 rows=55 loops=1) -> Nested loop inner join (cost=6.13 rows=10) (actual time=0.033..0.061 rows=10 loops=1) -> Covering index lookup on Matches using PRIMARY (Customer_Id=7) (cost=2.63 rows=10) (actual time=0.020..0.023 rows=10 loops=1) -> Single-row index lookup on c using PRIMARY (Color_Name=Matches.Color_Name) (cost=0.26 rows=1) (actual time=0.003..0.003 rows=1 loops=10) -> Filter: (cl.Brand is not null) (cost=5.46 rows=21) (actual time=0.021..0.025 rows=6 loops=10) -> Index lookup on cl using Clothing_Color (Clothing_Color=Matches.Color_Name) (cost=5.46 rows=21) (actual time=0.021..0.024 rows=6 loops=10) -> Filter: (b.Sustainability >= 65) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=55) -> Single-row index lookup on b using PRIMARY (Brand_Name=cl.Brand) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=55)

After Type Indexing

-> Limit: 15 row(s) (actual time=0.844..0.848 rows=15 loops=1) -> Sort: cl.Price, limit input to 15 row(s) per chunk (actual time=0.843..0.846 rows=15 loops=1) -> Stream results (cost=104.39 rows=70) (actual time=0.077..0.810 rows=55 loops=1) -> Nested loop inner join (cost=104.39 rows=70) (actual time=0.073..0.744 rows=55 loops=1) -> Nested loop inner join (cost=6.36 rows=13) (actual time=0.051..0.140 rows=40 loops=1) -> Nested loop inner join (cost=2.53 rows=13) (actual time=0.037..0.063 rows=40 loops=1) -> Filter: (b.Sustainability >= 65) (cost=0.65 rows=1) (actual time=0.025..0.028 rows=4 loops=1) -> Table scan on b (cost=0.65 rows=4) (actual time=0.023..0.025 rows=4 loops=1) -> Covering index lookup on Matches using PRIMARY (Customer_Id=7) (cost=1.16 rows=10) (actual time=0.006..0.008 rows=10 loops=4) -> Single-row index lookup on c using PRIMARY (Color_Name=Matches.Color_Name) (cost=0.20 rows=1) (actual time=0.002..0.002 rows=1 loops=40) -> Filter: (cl.Brand = b.Brand_Name) (cost=5.29 rows=5) (actual time=0.012..0.015 rows=1 loops=40) -> Index lookup on cl using Clothing_Color (Clothing_Color=Matches.Color_Name) (cost=5.29 rows=21) (actual time=0.011..0.014 rows=6 loops=40)

Indexing for Query #4

Original
-> Limit: 15 row(s) (actual time=44.412..44.415 rows=15 loops=1) -> Sort: Review_Count DESC (actual time=44.412..44.413 rows=15 loops=1) -> Filter: ((count(distinct Brands.Brand_Name) >= 2) and (count(Reviews.Review_Id) > 5)) (actual time=39.315..44.369 rows=67 loops=1) -> Stream results (actual time=39.068..44.100 rows=2225 loops=1) -> Group aggregate: count(Reviews.Review_Id), count(distinct Brands.Brand_Name), count(Reviews.Review_Id) (actual time=39.063..42.820 rows=2225 loops=1) -> Sort: c.Customer_Id, c.Email (actual time=39.048..39.905 rows=5534 loops=1) -> Stream results (cost=1607.90 rows=975) (actual time=0.277..29.024 rows=5534 loops=1) -> Nested loop inner join (cost=1607.90 rows=975) (actual time=0.274..25.818 rows=5534 loops=1) -> Nested loop inner join (cost=1266.65 rows=975) (actual time=0.262..16.753 rows=5534 loops=1) -> Covering index scan on b using PRIMARY (cost=0.65 rows=4) (actual time=0.029..0.041 rows=4 loops=1) -> Filter: ((r.Fit = 'Perfect') and (r.Customer_Id is not null)) (cost=78.84 rows=244) (actual time=0.163..4.071 rows=1384 loops=4) -> Index lookup on r using Brand (Brand=b.Brand_Name) (cost=78.84 rows=2438) (actual time=0.161..3.640 rows=2540 loops=4) -> Single-row index lookup on c using PRIMARY (Customer_Id=r.Customer_Id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5534)
After Fit Indexing
-> Limit: 15 row(s) (actual time=44.484..44.487 rows=15 loops=1) -> Sort: Review_Count DESC (actual time=44.484..44.485 rows=15 loops=1) -> Filter: ((count(distinct Brands.Brand_Name) >= 2) and (count(Reviews.Review_Id) > 5)) (actual time=40.081..44.449 rows=67 loops=1) -> Stream results (actual time=39.779..44.224 rows=2225 loops=1) -> Group aggregate: count(Reviews.Review_Id), count(distinct Brands.Brand_Name), count(Reviews.Review_Id) (actual time=39.775..43.129 rows=2225 loops=1) -> Sort: c.Customer_Id, c.Email (actual time=39.755..40.508 rows=5534 loops=1) -> Stream results (cost=3203.55 rows=5534) (actual time=0.281..29.320 rows=5534 loops=1) -> Nested loop inner join (cost=3203.55 rows=5534) (actual time=0.278..26.032 rows=5534 loops=1) -> Nested loop inner join (cost=1266.65 rows=5534) (actual time=0.265..16.964 rows=5534 loops=1) -> Covering index scan on b using PRIMARY (cost=0.65 rows=4) (actual time=0.032..0.045 rows=4 loops=1) -> Filter: ((r.Fit = 'Perfect') and (r.Customer_Id is not null)) (cost=107.34 rows=1384) (actual time=0.174..4.129 rows=1384 loops=4) -> Index lookup on r using Brand (Brand=b.Brand_Name) (cost=107.34 rows=2438) (actual time=0.172..3.704 rows=2540 loops=4) -> Single-row index lookup on c using PRIMARY (Customer_Id=r.Customer_Id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5534)
After First_Name Indexing
-> Limit: 15 row(s) (actual time=45.593..45.595 rows=15 loops=1) -> Sort: Review_Count DESC (actual time=45.592..45.594 rows=15 loops=1) -> Filter: ((count(distinct Brands.Brand_Name) >= 2) and (count(Reviews.Review_Id) > 5)) (actual time=41.197..45.560 rows=67 loops=1) -> Stream results (actual time=40.924..45.332 rows=2225 loops=1) -> Group aggregate: count(Reviews.Review_Id), count(distinct Brands.Brand_Name), count(Reviews.Review_Id) (actual time=40.918..44.251 rows=2225 loops=1) -> Sort: c.Customer_Id, c.Email (actual time=40.896..41.663 rows=5534 loops=1) -> Stream results (cost=1607.90 rows=975) (actual time=0.364..29.939 rows=5534 loops=1) -> Nested loop inner join (cost=1607.90 rows=975) (actual time=0.361..26.640 rows=5534 loops=1) -> Nested loop inner join (cost=1266.65 rows=975) (actual

time=0.346..17.400 rows=5534 loops=1) -> Covering index scan on b using PRIMARY (cost=0.65 rows=4) (actual time=0.037..0.049 rows=4 loops=1) -> Filter: ((r.Fit = 'Perfect') and (r.Customer_Id is not null)) (cost=78.84 rows=244) (actual time=0.184..4.212 rows=1384 loops=4) -> Index lookup on r using Brand (Brand=b.Brand_Name) (cost=78.84 rows=2438) (actual time=0.183..3.765 rows=2540 loops=4) -> Single-row index lookup on c using PRIMARY (Customer_Id=r.Customer_Id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5534)

After Last_Name Indexing

> Limit: 15 row(s) (actual time=44.952..44.954 rows=15 loops=1) -> Sort: Review_Count DESC (actual time=44.951..44.953 rows=15 loops=1) -> Filter: ((count(distinct Brands.Brand_Name) >= 2) and (count(Reviews.Review_Id) > 5)) (actual time=40.607..44.920 rows=67 loops=1) -> Stream results (actual time=40.348..44.704 rows=2225 loops=1) -> Group aggregate: count(Reviews.Review_Id), count(distinct Brands.Brand_Name), count(Reviews.Review_Id) (actual time=40.342..43.630 rows=2225 loops=1) -> Sort: c.Customer_Id, c.Email (actual time=40.302..41.055 rows=5534 loops=1) -> Stream results (cost=1607.90 rows=975) (actual time=0.230..29.130 rows=5534 loops=1) -> Nested loop inner join (cost=1607.90 rows=975) (actual time=0.228..25.982 rows=5534 loops=1) -> Nested loop inner join (cost=1266.65 rows=975) (actual time=0.218..16.789 rows=5534 loops=1) -> Covering index scan on b using PRIMARY (cost=0.65 rows=4) (actual time=0.015..0.028 rows=4 loops=1) -> Filter: ((r.Fit = 'Perfect') and (r.Customer_Id is not null)) (cost=78.84 rows=244) (actual time=0.164..4.090 rows=1384 loops=4) -> Index lookup on r using Brand (Brand=b.Brand_Name) (cost=78.84 rows=2438) (actual time=0.162..3.661 rows=2540 loops=4) -> Single-row index lookup on c using PRIMARY (Customer_Id=r.Customer_Id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5534)