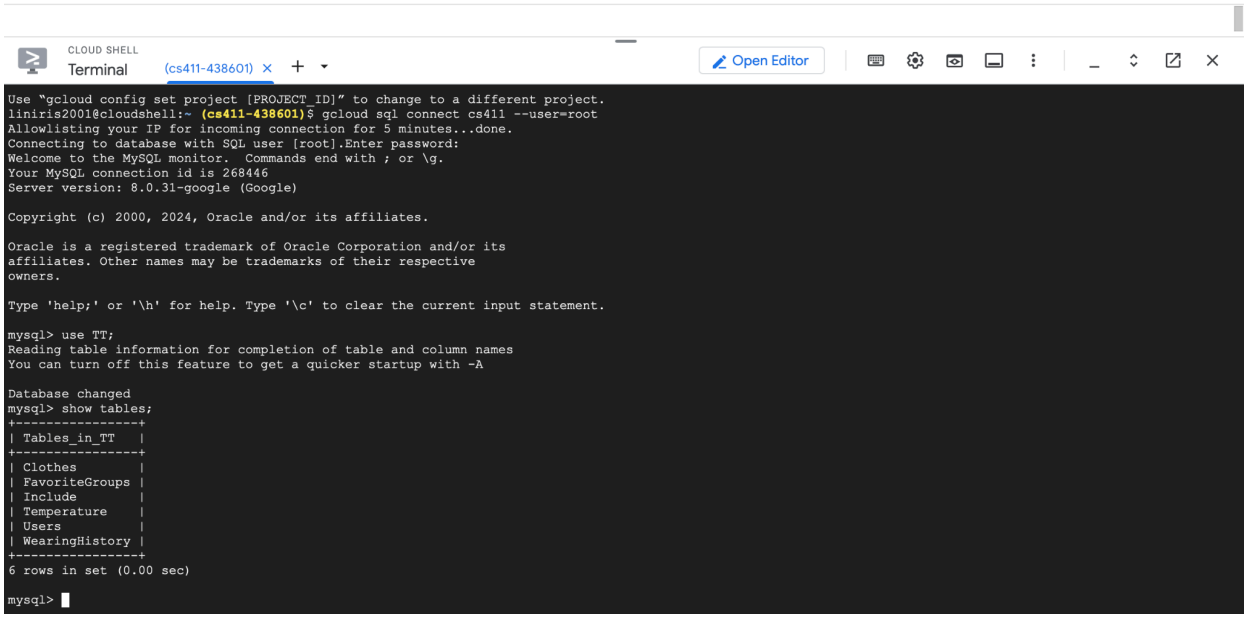
Stage 3: Database Implementation and Indexing

Team number: 33
Team name: TT
Team member:
Supawit Sutthiboriban (supawit3)
Yu-Ting Cheng (ytcheng4)
Yu-Chien Lin (yuchien4)
Han-Chih Chang (hanchih2)

Database implementation

1. **GCP Connection**

   Database name: TT



2. **DDL Commands**

```
CREATE TABLE Users(
    UserId INT PRIMARY KEY,
    FirstName VARCHAR(225) NOT NULL,
    LastName VARCHAR(225) NOT NULL,
    PhoneNumber REAL,
    Email VARCHAR(225) NOT NULL,
    Password VARCHAR(225) NOT NULL
);

CREATE TABLE Temperature (
```

```sql
    TemperatureLevel INT PRIMARY KEY,
    TemperatureMin INT NOT NULL,
    TemperatureMax INT NOT NULL
);

CREATE TABLE Clothes (
    ClothId INT PRIMARY KEY,
    UserId INT NOT NULL,
    ClothName VARCHAR(255) NOT NULL,
    Category VARCHAR(255) NOT NULL,
    Subcategory VARCHAR(255) NOT NULL,
    Color VARCHAR(255) NOT NULL,
    Usages VARCHAR(255),
    Image VARCHAR(255),
    TemperatureLevel INT,
    FOREIGN KEY (UserId) REFERENCES Users(UserId) ON DELETE CASCADE,
    FOREIGN KEY (TemperatureLevel) REFERENCES Temperature(TemperatureLevel)
ON        DELETE SET NULL
);

CREATE TABLE FavoriteGroups (
    FavoriteId INT PRIMARY KEY,
    GroupName VARCHAR(255) NOT NULL,
    UserId INT NOT NULL,
    FOREIGN KEY (UserId) REFERENCES Users(UserId) ON DELETE CASCADE
);

CREATE TABLE Include (
    FavoriteId INT NOT NULL,
    ClothId INT NOT NULL,
    PRIMARY KEY (FavoriteId, ClothId),
    FOREIGN KEY (FavoriteId) REFERENCES FavoriteGroups(FavoriteId) ON
DELETE CASCADE,
    FOREIGN KEY (ClothId) REFERENCES Clothes(ClothId) ON DELETE CASCADE
);

CREATE TABLE WearingHistory (
    Date DATE NOT NULL,
    UserId INT NOT NULL,
    Cloth1 INT,
    Cloth2 INT,
    Cloth3 INT,
    Cloth4 INT,
```

```
    Cloth5 INT,
    PRIMARY KEY (Date, UserId),
    FOREIGN KEY (UserId) REFERENCES Users(UserId) ON DELETE CASCADE,
    FOREIGN KEY (Cloth1) REFERENCES Clothes(ClothId) ON DELETE SET NULL,
    FOREIGN KEY (Cloth2) REFERENCES Clothes(ClothId) ON DELETE SET NULL,
    FOREIGN KEY (Cloth3) REFERENCES Clothes(ClothId) ON DELETE SET NULL,
    FOREIGN KEY (Cloth4) REFERENCES Clothes(ClothId) ON DELETE SET NULL,
    FOREIGN KEY (Cloth5) REFERENCES Clothes(ClothId) ON DELETE SET NULL
);
```

3. **Data records**
   a. Table name: Users

```
mysql> select count(*) from Users;
+----------+
| count(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.01 sec)

mysql> describe Users;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| UserId      | int          | NO   | PRI | NULL    |       |
| FirstName   | varchar(225) | NO   |     | NULL    |       |
| LastName    | varchar(225) | NO   |     | NULL    |       |
| PhoneNumber | double       | YES  |     | NULL    |       |
| Email       | varchar(225) | NO   |     | NULL    |       |
| Password    | varchar(225) | NO   |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

   b. Table name: Clothes

```
mysql> select count(*) from Clothes;
+----------+
| count(*) |
+----------+
|    15911 |
+----------+
1 row in set (0.00 sec)

mysql> describe Clothes;
+------------------+--------------+------+-----+---------+-------+
| Field            | Type         | Null | Key | Default | Extra |
+------------------+--------------+------+-----+---------+-------+
| ClothId          | int          | NO   | PRI | NULL    |       |
| UserId           | int          | NO   | MUL | NULL    |       |
| ClothName        | varchar(255) | NO   |     | NULL    |       |
| Category         | varchar(255) | NO   |     | NULL    |       |
| Subcategory      | varchar(255) | NO   |     | NULL    |       |
| Color            | varchar(255) | NO   |     | NULL    |       |
| Usages           | varchar(255) | YES  |     | NULL    |       |
| Image            | varchar(255) | YES  |     | NULL    |       |
| TemperatureLevel | int          | YES  | MUL | NULL    |       |
+------------------+--------------+------+-----+---------+-------+
9 rows in set (0.00 sec)
```

c. Table name: Temperature

```
mysql> select count(*) from Temperature;
+----------+
| count(*) |
+----------+
|        4 |
+----------+
1 row in set (0.01 sec)

mysql> describe Temperature;
+------------------+------+------+-----+---------+-------+
| Field            | Type | Null | Key | Default | Extra |
+------------------+------+------+-----+---------+-------+
| TemperatureLevel | int  | NO   | PRI | NULL    |       |
| TemperatureMin   | int  | NO   |     | NULL    |       |
| TemperatureMax   | int  | NO   |     | NULL    |       |
+------------------+------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

d. Table name: FavoriteGroups

```
mysql> select count(*) from FavoriteGroups;
+----------+
| count(*) |
+----------+
|     1503 |
+----------+
1 row in set (0.01 sec)

mysql> describe FavoriteGroups;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| FavoriteId | int          | NO   | PRI | NULL    |       |
| GroupName  | varchar(255) | NO   |     | NULL    |       |
| UserId     | int          | NO   | MUL | NULL    |       |
+------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

e. Table name: Include

```
mysql> select count(*) from Include;
+----------+
| count(*) |
+----------+
|     5436 |
+----------+
1 row in set (0.00 sec)

mysql> describe Include;
+------------+------+------+-----+---------+-------+
| Field      | Type | Null | Key | Default | Extra |
+------------+------+------+-----+---------+-------+
| FavoriteId | int  | NO   | PRI | NULL    |       |
| ClothId    | int  | NO   | PRI | NULL    |       |
+------------+------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

f. Table name: WearingHistory

```
mysql> select count(*) from WearingHistory;
+----------+
| count(*) |
+----------+
|     8000 |
+----------+
1 row in set (0.00 sec)

mysql> describe WearingHistory;;
+--------+------+------+-----+---------+-------+
| Field  | Type | Null | Key | Default | Extra |
+--------+------+------+-----+---------+-------+
| Date   | date | NO   | PRI | NULL    |       |
| UserId | int  | NO   | PRI | NULL    |       |
| Cloth1 | int  | YES  | MUL | NULL    |       |
| Cloth2 | int  | YES  | MUL | NULL    |       |
| Cloth3 | int  | YES  | MUL | NULL    |       |
| Cloth4 | int  | YES  | MUL | NULL    |       |
| Cloth5 | int  | YES  | MUL | NULL    |       |
+--------+------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

Advanced Queries

1 .Query: Summarize the frequency of cool-weather outfit (Topwear and Bottomwear) worn in the past two weeks

```
SELECT c.ClothName, COUNT(*) AS Frequency
FROM WearingHistory wh
```

```sql
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR
wh.Cloth3 = c.ClothId
                OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
JOIN Temperature t ON t.TemperatureLevel = c.TemperatureLevel
WHERE wh.UserId = 1 AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14
DAY) AND '2024-10-25' AND c.Category IN ('Topwear', 'Bottomwear')
AND t.TemperatureMin <= 50
GROUP BY c.ClothName
ORDER BY Frequency DESC;
```

Example with UserId = 1 and End date = ‘2024-10-25’

```
mysql> SELECT c.ClothName, COUNT(*) AS Frequency
    -> FROM WearingHistory wh
    -> JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR wh.Cloth3 = c.ClothId
    ->                OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
    -> JOIN Temperature t ON t.TemperatureLevel = c.TemperatureLevel
    -> WHERE wh.UserId = 1 AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND '2024-10-25' AND c.Category IN ('Topwear', 'Bottomwear')
    -> AND t.TemperatureMin <= 50
    -> GROUP BY c.ClothName
    -> ORDER BY Frequency DESC;
+---------------------------------------------+-----------+
| ClothName                                   | Frequency |
+---------------------------------------------+-----------+
| United Colors of Benetton Women Black Trouser |         3 |
| DUSG Women Uttarkashi Organic Endive Top      |         2 |
| PUMA Women Yellow Printed T-shirt             |         2 |
| Puma Women's Powergirls T-shirt               |         1 |
+---------------------------------------------+-----------+
4 rows in set (0.02 sec)
```

(The output is less than 15 rows)

2. Query: Display the frequency of ‘Topwear’ clothing colors worn by the user within one month

```sql
SELECT c.Color, COUNT(*) AS ColorFrequency
FROM WearingHistory wh
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR
wh.Cloth3 = c.ClothId
                OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
WHERE wh.UserId = 123 AND c.Category = 'Topwear'
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 30 DAY) AND
'2024-10-25'
  AND c.Color IS NOT NULL
GROUP BY c.Color
ORDER BY ColorFrequency DESC;
```

Example with UserId = 123 and End date = '2024-10-25' and Category = 'Topwear'

```
mysql> SELECT c.Color, COUNT(*) AS ColorFrequency
    -> FROM WearingHistory wh
    -> JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR wh.Cloth3 = c.ClothId
    ->                OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
    -> WHERE wh.UserId = 123 AND c.Category = 'Topwear'
    ->   AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 30 DAY) AND '2024-10-25'
    ->   AND c.Color IS NOT NULL
    -> GROUP BY c.Color
    -> ORDER BY ColorFrequency DESC;
+-----------+----------------+
| Color     | ColorFrequency |
+-----------+----------------+
| Navy Blue |              4 |
| White     |              2 |
+-----------+----------------+
2 rows in set (0.01 sec)
```

(The output is less than 15 rows)

3. Query: List items that are not black, not for sports, and in one of the FavoriteGroups that were worn in the latest 2 weeks

```
SELECT c.ClothName, fg.GroupName, c.Color, c.Usages
FROM Clothes c
JOIN Include i ON c.ClothId = i.ClothId
JOIN FavoriteGroups fg ON i.FavoriteId = fg.FavoriteId
JOIN WearingHistory wh ON (c.ClothId = wh.Cloth1 OR c.ClothId = wh.Cloth2
                           OR c.ClothId = wh.Cloth3 OR c.ClothId = wh.Cloth4
                           OR c.ClothId = wh.Cloth5)
WHERE c.Color != 'black'
  AND c.Usages != 'sports'
  AND fg.UserId = 2
  AND wh.UserId = 2
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND '2024-10-25'
GROUP BY fg.GroupName, c.ClothName, c.Color, c.Usages;
```

Example with UserId = 2 and End date = ‘2024-10-25’

```
mysql> SELECT c.ClothName, fg.GroupName, c.Color, c.Usages
    -> FROM Clothes c
    -> JOIN Include i ON c.ClothId = i.ClothId
    -> JOIN FavoriteGroups fg ON i.FavoriteId = fg.FavoriteId
    -> JOIN WearingHistory wh ON (c.ClothId = wh.Cloth1 OR c.ClothId = wh.Cloth2
    ->                            OR c.ClothId = wh.Cloth3 OR c.ClothId = wh.Cloth4
    ->                            OR c.ClothId = wh.Cloth5)
    -> WHERE c.Color != 'black'
    ->   AND c.Usages != 'sports'
    ->   AND fg.UserId = 2
    ->   AND wh.UserId = 2
    ->   AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND '2024-10-25'
    -> GROUP BY fg.GroupName, c.ClothName, c.Color, c.Usages;
+--------------------------------------------------+-----------+-------+--------+
| ClothName                                        | GroupName | Color | Usages |
+--------------------------------------------------+-----------+-------+--------+
| Puma Women Corsica Tribal White Casual Shoes     | Mrprd     | White | Casual |
| Gini and Jony Girl's Scotia White Purple Kidswear | Glkyptzb  | White | Casual |
| Catwalk Women Brown Wedges                       | Npvvtxn   | Brown | Casual |
| Forever New Women Blossom  Pink Printed Skirt    | Mrprd     | Pink  | Casual |
| Rocia Women Casual Tan Sandal                    | Mrprd     | Tan   | Casual |
+--------------------------------------------------+-----------+-------+--------+
5 rows in set (0.00 sec)
```

(The output is less than 15 rows)

4. Query: List ClothId of some ‘’Category” that were often worn (time > threshold within the last two weeks) for certain ‘Usages’ but not in any favorite group

```
(
SELECT ClothId
FROM WearingHistory wh
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR
wh.Cloth3 = c.ClothId
               OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
WHERE wh.UserId = 120 AND c.Usages = 'Casual' AND c.Category = 'Topwear'
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND
'2024-10-25'
  AND c.ClothId IS NOT NULL
GROUP BY c.ClothId
HAVING COUNT(*) >= 3
) EXCEPT (
SELECT ClothId
FROM Include NATURAL JOIN FavoriteGroups NATURAL JOIN Clothes
WHERE UserId = 120
);
```

Example with UserId = 120, EndDate = ‘2024-10-25’, Threshold=3, Usages = ‘Casual’, Category = ’Topwear’

```
mysql> (
    -> SELECT ClothId
    -> FROM WearingHistory wh
    -> JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR wh.Cloth3 = c.ClothId
    ->                OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
    -> WHERE wh.UserId = 120 AND c.Usages = 'Casual' AND c.Category = 'Topwear'
    ->   AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND '2024-10-25'
    ->   AND c.ClothId IS NOT NULL
    -> GROUP BY c.ClothId
    -> HAVING COUNT(*) >= 3
    -> ) EXCEPT (
    -> SELECT ClothId
    -> FROM Include NATURAL JOIN FavoriteGroups NATURAL JOIN Clothes
    -> WHERE UserId = 120
    -> );
+---------+
| ClothId |
+---------+
|    1897 |
|    1898 |
+---------+
2 rows in set (0.02 sec)
```

(The output is less than 15 rows)

Indexing Analysis Reports

1. **Query: Summarize the frequency of cool-weather outfit worn in the past two weeks**

Option 0: Baseline

```
EXPLAIN ANALYZE SELECT c.ClothName, COUNT(*) AS Frequency
FROM WearingHistory wh
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR
wh.Cloth3 = c.ClothId
                OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
JOIN Temperature t ON t.TemperatureLevel = c.TemperatureLevel
WHERE wh.UserId = 1 AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14
DAY) AND '2024-10-25' AND c.Category IN ('Topwear', 'Bottomwear')
AND t.TemperatureMin <= 50
GROUP BY c.ClothName
ORDER BY Frequency DESC;
```

```
mysql> EXPLAIN ANALYZE SELECT c.ClothName, COUNT(*) AS Frequency
    -> FROM WearingHistory wh
    -> JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR wh.Cloth3 = c.ClothId
    ->                OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
    -> JOIN Temperature t ON t.TemperatureLevel = c.TemperatureLevel
    -> WHERE wh.UserId = 1 AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND '2024-10-25' AND c.Category IN ('Topwear', 'Bottomwear')
    -> AND t.TemperatureMin <= 50
    -> GROUP BY c.ClothName
    -> ORDER BY Frequency DESC;
```

```
| -> Sort: Frequency DESC  (actual time=20.633..20.634 rows=4 loops=1)
   -> Table scan on <temporary>  (actual time=20.605..20.606 rows=4 loops=1)
       -> Aggregate using temporary table  (actual time=20.603..20.603 rows=4 loops=1)
           -> Nested loop inner join  (cost=1001.67 rows=0.3) (actual time=3.352..20.495 rows=8 loops=1)
               -> Inner hash join (no condition)  (cost=3.92 rows=1) (actual time=0.296..0.353 rows=18 loops=1)
                   -> Filter: (t.TemperatureMin <= 50)  (cost=0.22 rows=1) (actual time=0.051..0.076 rows=3 loops=1)
                       -> Table scan on t  (cost=0.22 rows=4) (actual time=0.048..0.069 rows=4 loops=1)
                   -> Hash
                       -> Filter: ((wh.UserId = 1) and (wh.`Date` between <cache>(('2024-10-25' - interval 14 day)) and '2024-10-25'))  (cost=3.00 rows=3) (actual time=0.153..0.206 rows=6 loops=1)
                           -> Intersect rows sorted by row ID  (cost=3.00 rows=3) (actual time=0.132..0.181 rows=6 loops=1)
                               -> Index range scan on wh using UserId over (UserId = 1 AND '2024-10-11' <= Date <= '2024-10-25')  (cost=0.86 rows=6) (actual time=0.094..0.109 rows=6 loops=1)
               -> Filter: (c.Category in ('Topwear','Bottomwear'))  (cost=120.76 rows=0.3) (actual time=1.117..1.118 rows=0 loops=18)
                   -> Index lookup on c using TemperatureLevel (TemperatureLevel=t.TemperatureLevel), with index condition: ((c.ClothId = wh.Cloth1) or (c.ClothId = wh.Cloth2) or (c.ClothId = wh.Cloth3) or (c.ClothId = wh.Cloth4) or (c.Clot
hId = wh.Cloth5))  (cost=120.76 rows=3862) (actual time=1.115..1.116 rows=1 loops=18)
 |
```

Based on the result, we found two attributes in the execution that have not been indexed (since

they are neither primary keys nor foreign keys): t.TemperatureMin and c.Category. A table scan cost for t.TemperatureMin is 0.22 and a table lookup cost for c.Category is 120.76. Therefore, we decided to perform indexings based on these two attributes.

Option 1: c.Category Indexing

```
create index catg_idx on Clothes(Category)
EXPLAIN ANALYZE SELECT c.ClothName, COUNT(*) AS Frequency
FROM WearingHistory wh
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR
wh.Cloth3 = c.ClothId
                OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
JOIN Temperature t ON t.TemperatureLevel = c.TemperatureLevel
WHERE wh.UserId = 1 AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14
DAY) AND '2024-10-25' AND c.Category IN ('Topwear', 'Bottomwear')
AND t.TemperatureMin <= 50
GROUP BY c.ClothName
ORDER BY Frequency DESC;
```

```
| -> Sort: Frequency DESC  (actual time=35.155..35.155 rows=4 loops=1)
    -> Table scan on <temporary>  (actual time=35.129..35.131 rows=4 loops=1)
        -> Aggregate using temporary table  (actual time=35.124..35.124 rows=4 loops=1)
            -> Nested loop inner join  (cost=1001.67 rows=1)  (actual time=4.996..35.007 rows=8 loops=1)
                -> Inner hash join (no condition)  (cost=3.92 rows=1)  (actual time=0.180..0.233 rows=18 loops=1)
                    -> Filter: (t.TemperatureMin <= 50)  (cost=0.22 rows=1)  (actual time=0.012..0.026 rows=3 loops=1)
                        -> Table scan on t  (cost=0.22 rows=4)  (actual time=0.011..0.021 rows=4 loops=1)
                    -> Hash
                        -> Filter: ((wh.UserId = 1) and (wh.`Date` between <cache>((‘2024-10-25’ - interval 14 day)) and ‘2024-10-25’))  (cost=3.00 rows=3)  (actual time=0.106..0.154 rows=6 loops=1)
                            -> Intersect rows sorted by row ID  (cost=3.00 rows=3)  (actual time=0.097..0.140 rows=6 loops=1)
                                -> Index range scan on wh using UserId over (UserId = 1 AND ‘2024-10-11’ <= Date <= ‘2024-10-25’)  (cost=0.86 rows=6)  (actual time=0.053..0.065 rows=6 loops=1)
                -> Filter: (c.Category in (‘Topwear’,’Bottomwear’))  (cost=120.77 rows=1)  (actual time=1.929..1.931 rows=0 loops=18)
                    -> Index lookup on c using TemperatureLevel (TemperatureLevel=t.TemperatureLevel), with index condition: ((c.ClothId = wh.Cloth1) or (c.ClothId = wh.Cloth2) or (c.ClothId = wh.Cloth3) or (c.ClothId = wh.Cloth4) or (c.Clot
hId = wh.Cloth5))  (cost=120.77 rows=3862)  (actual time=1.921..1.927 rows=1 loops=18)
 |
```

First, we tried indexing c.Category. However, we found that this indexing strategy didn’t improve query performance, as the cost of the index lookup remained about the same at 120.77. We believe this is because indexing may not always improve performance for range searches on categorical columns, particularly when the column has only a few distinct values like this, since many rows still need to be accessed.

Option 2: c.Category and t.TemperatureMin Indexing

*note: we already have catg_index from option 1

```
create index min_idx on Temperature(TemperatureMin);
EXPLAIN ANALYZE SELECT c.ClothName, COUNT(*) AS Frequency
FROM WearingHistory wh
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR
wh.Cloth3 = c.ClothId
                OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
JOIN Temperature t ON t.TemperatureLevel = c.TemperatureLevel
WHERE wh.UserId = 1 AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14
DAY) AND '2024-10-25' AND c.Category IN ('Topwear', 'Bottomwear')
AND t.TemperatureMin <= 50
GROUP BY c.ClothName
ORDER BY Frequency DESC;
```

```
| -> Sort: Frequency DESC  (actual time=26.847..26.848 rows=4 loops=1)
    -> Table scan on <temporary>  (actual time=26.820..26.821 rows=4 loops=1)
        -> Aggregate using temporary table  (actual time=26.816..26.816 rows=4 loops=1)
            -> Nested loop inner join  (cost=795.72 rows=0.001) (actual time=0.209..26.787 rows=8 loops=1)
                -> Filter: ((c.ClothId = wh.Cloth1) or (c.ClothId = wh.Cloth2) or (c.ClothId = wh.Cloth3) or (c.ClothId = wh.Cloth4) or (c.ClothId = wh.Cloth5))  (cost=795.08 rows=0.001) (actual time=0.198..26.766 rows=12 loops=1)
                    -> Inner hash join (no condition)  (cost=795.08 rows=0.001) (actual time=0.171..16.694 rows=47634 loops=1)
                        -> Filter: ((c.Category in ('Topwear','Bottomwear')) and (c.TemperatureLevel is not null))  (cost=263.85 rows=3) (actual time=0.039..11.830 rows=7939 loops=1)
                            -> Table scan on c  (cost=263.85 rows=15447) (actual time=0.035..6.833 rows=15911 loops=1)
                        -> Hash
                            -> Filter: ((wh.UserId = 1) and (wh.`Date` between <cache>(('2024-10-25' - interval 14 day)) and '2024-10-25'))  (cost=3.00 rows=3) (actual time=0.081..0.117 rows=6 loops=1)
                                -> Intersect rows sorted by row ID  (cost=3.00 rows=3) (actual time=0.069..0.102 rows=6 loops=1)
                                    -> Index range scan on wh using UserId over (UserId = 1 AND '2024-10-11' <= Date <= '2024-10-25')  (cost=0.86 rows=6) (actual time=0.055..0.064 rows=6 loops=1)
                -> Filter: (t.TemperatureMin <= 50)  (cost=0.09 rows=1) (actual time=0.001..0.001 rows=1 loops=12)
                    -> Single-row index lookup on t using PRIMARY (TemperatureLevel=c.TemperatureLevel)  (cost=0.09 rows=1) (actual time=0.001..0.001 rows=1 loops=12)
|
```

We tried indexing t.TemperatureMin on top of our first option. This addition completely changed the query plan. As for t.TemperatureMin itself, the cost of index lookup decreased from 0.22 to 0.09. However, for c.Category, the query plan shifted to a full table scan instead of an index lookup, which increased the cost from 120.77 to 263.85. Despite this, the nested loop inner join cost improved from 1001.97 to 795.92. We believe this is because the selectivity of the c.Category column is low, as we mentioned earlier, making the full table scan more efficient in this case than using the index lookup. Therefore, we believe that option 2 is better than option 1 and baseline.

Option 3 t.TemperatureMin indexing

```
drop index catg_idx on Clothes;
EXPLAIN ANALYZE SELECT c.ClothName, COUNT(*) AS Frequency
FROM WearingHistory wh
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR
wh.Cloth3 = c.ClothId
                OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
JOIN Temperature t ON t.TemperatureLevel = c.TemperatureLevel
WHERE wh.UserId = 1 AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14
DAY) AND '2024-10-25' AND c.Category IN ('Topwear', 'Bottomwear')
AND t.TemperatureMin <= 50
GROUP BY c.ClothName
ORDER BY Frequency DESC;
```

```
| -> Sort: Frequency DESC  (actual time=46.353..46.353 rows=4 loops=1)
    -> Table scan on <temporary>  (actual time=46.324..46.325 rows=4 loops=1)
        -> Aggregate using temporary table  (actual time=46.319..46.319 rows=4 loops=1)
            -> Nested loop inner join  (cost=1279.97 rows=0.0001) (actual time=0.262..46.283 rows=8 loops=1)
                -> Filter: ((c.ClothId = wh.Cloth1) or (c.ClothId = wh.Cloth2) or (c.ClothId = wh.Cloth3) or (c.ClothId = wh.Cloth4) or (c.ClothId = wh.Cloth5))  (cost=1279.72 rows=0.0002) (actual time=0.250..46.255 rows=12 loops=1)
                    -> Inner hash join (no condition)  (cost=1279.72 rows=0.0002) (actual time=0.204..29.246 rows=47634 loops=1)
                        -> Filter: ((c.Category in ('Topwear','Bottomwear')) and (c.TemperatureLevel is not null))  (cost=425.51 rows=1) (actual time=0.063..20.958 rows=7939 loops=1)
                            -> Table scan on c  (cost=425.51 rows=15447) (actual time=0.058..12.093 rows=15911 loops=1)
                        -> Hash
                            -> Filter: ((wh.UserId = 1) and (wh.`Date` between <cache>(('2024-10-25' - interval 14 day)) and '2024-10-25'))  (cost=3.00 rows=3) (actual time=0.076..0.125 rows=6 loops=1)
                                -> Intersect rows sorted by row ID  (cost=3.00 rows=3) (actual time=0.065..0.110 rows=6 loops=1)
                                    -> Index range scan on wh using UserId over (UserId = 1 AND '2024-10-11' <= Date <= '2024-10-25')  (cost=0.86 rows=6) (actual time=0.049..0.062 rows=6 loops=1)
                -> Filter: (t.TemperatureMin <= 50)  (cost=0.11 rows=1) (actual time=0.002..0.002 rows=1 loops=12)
                    -> Single-row index lookup on t using PRIMARY (TemperatureLevel=c.TemperatureLevel)  (cost=0.11 rows=1) (actual time=0.001..0.002 rows=1 loops=12)
|
```

Lastly, we dropped the c.Category index, leaving t.TemperatureMin as a single index. The cost for the index lookup on t.TemperatureMin remains around the same level as option 2 at 0.11. However, removing the c.Category index significantly increased the table scan cost to 425.51. Also, the nested loop inner join cost rose to 1279.72. This suggests that indexing c.Category is beneficial for table c scanning and nested loop inner join, and therefore we should keep it.

Conclusion: Option 2: c.Category and t.TemperatureMin Indexing is the best

2. Query: Display the frequency of ‘Topwear’ clothing colors worn by the user within one month

Option 0 baseline

```sql
explain analyze SELECT c.Color, COUNT(*) AS ColorFrequency
FROM WearingHistory wh
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR
wh.Cloth3 = c.ClothId
                OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
WHERE wh.UserId = 123 AND c.Category = 'Topwear'
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 30 DAY) AND
'2024-10-25'
  AND c.Color IS NOT NULL
GROUP BY c.Color
ORDER BY ColorFrequency DESC;
```

```
| -> Sort: ColorFrequency DESC  (actual time=0.342..0.342 rows=2 loops=1)
    -> Table scan on <temporary>  (actual time=0.328..0.329 rows=2 loops=1)
        -> Aggregate using temporary table  (actual time=0.327..0.327 rows=2 loops=1)
            -> Nested loop inner join  (cost=6066.63 rows=3) (actual time=0.122..0.301 rows=6 loops=1)
                -> Filter: ((wh.UserId = 123) and (wh.`Date` between <cache>(('2024-10-25' - interval 30 day)) and '2024-10-25'))  (cost=1.59 rows=3) (actual time=0.064..0.093 rows=6
loops=1)
                    -> Intersect rows sorted by row ID  (cost=1.59 rows=3) (actual time=0.057..0.083 rows=6 loops=1)
                        -> Index range scan on wh using UserId over (UserId = 123 AND '2024-09-25' <= Date <= '2024-10-25')  (cost=0.25 rows=6) (actual time=0.044..0.053 rows=6 loops=
1)
                -> Filter: ((c.Category = 'Topwear') and ((c.ClothId = wh.Cloth1) or (c.ClothId = wh.Cloth2) or (c.ClothId = wh.Cloth3) or (c.ClothId = wh.Cloth4) or (c.ClothId = wh.C
loth5)))  (cost=477.01 rows=1) (actual time=0.029..0.034 rows=1 loops=6)
                    -> Index range scan on c (re-planned for each iteration)  (cost=477.01 rows=15447) (actual time=0.028..0.032 rows=4 loops=6)
 |
```

Based on the result, we found two attributes in the execution that have not been indexed (since they are neither primary keys nor foreign keys): c.Color and c.Category. A table lookup cost for c.Category is 477.01. The cost related to the use of c.Color would be included in the temporary table aggregation and grouping steps, where ColorFrequency is calculated, but there isn't a specific cost component attributed solely to c.Color, so we would use the cost of Nested loop inner join to evaluate the cost of c.Color, which is 6066.63. In summary, we decided to perform indexings based on these two attributes.

Option 1 Color_idx

```sql
create index Color_idx on Clothes(Color);
explain analyze SELECT c.Color, COUNT(*) AS ColorFrequency
FROM WearingHistory wh
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR
wh.Cloth3 = c.ClothId
                OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
WHERE wh.UserId = 123 AND c.Category = 'Topwear'
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 30 DAY) AND
'2024-10-25'
  AND c.Color IS NOT NULL
GROUP BY c.Color
ORDER BY ColorFrequency DESC;
```

```
| -> Sort: ColorFrequency DESC  (actual time=0.389..0.389 rows=2 loops=1)
    -> Table scan on <temporary>  (actual time=0.374..0.375 rows=2 loops=1)
        -> Aggregate using temporary table  (actual time=0.372..0.372 rows=2 loops=1)
            -> Nested loop inner join  (cost=6066.63 rows=3) (actual time=0.137..0.345 rows=6 loops=1)
                -> Filter: ((wh.UserId = 123) and (wh.`Date` between <cache>(('2024-10-25' - interval 30 day)) and '2024-10-25'))  (cost=1.59 rows=3) (actual time=0.035..0.063 rows=6
loops=1)
                    -> Intersect rows sorted by row ID  (cost=1.59 rows=3) (actual time=0.030..0.054 rows=6 loops=1)
                        -> Index range scan on wh using UserId over (UserId = 123 AND '2024-09-25' <= Date <= '2024-10-25')  (cost=0.25 rows=6) (actual time=0.021..0.029 rows=6 loops=
1)
                -> Filter: ((c.Category = 'Topwear') and ((c.ClothId = wh.Cloth1) or (c.ClothId = wh.Cloth2) or (c.ClothId = wh.Cloth3) or (c.ClothId = wh.Cloth4) or (c.ClothId = wh.C
loth5)))  (cost=477.01 rows=1) (actual time=0.041..0.046 rows=1 loops=6)
                    -> Index range scan on c (re-planned for each iteration)  (cost=477.01 rows=15447) (actual time=0.040..0.044 rows=4 loops=6)
 |
```

First, we tried indexing c.Color. However, we found that this indexing strategy didn't improve

query performance, as the cost of the index lookup remained about the same at 6066.63. We believe this is because of two reasons: (1) Low Selectivity: If Color values are repeated often, the index becomes less useful since scanning the index could be as costly as scanning the entire table. (2) Query/Optimizer Choice: If Color isn't frequently used in WHERE clauses, the optimizer may not select the index.

Option 2 Category_idx

```
Drop index Color_idx on Clothes;
create index Category_idx on Clothes(Category);
explain analyze SELECT c.Color, COUNT(*) AS ColorFrequency
FROM WearingHistory wh
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR
wh.Cloth3 = c.ClothId
               OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
WHERE wh.UserId = 123 AND c.Category = 'Topwear'
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 30 DAY) AND
'2024-10-25'
  AND c.Color IS NOT NULL
GROUP BY c.Color
ORDER BY ColorFrequency DESC;
```

```
| -> Sort: ColorFrequency DESC  (actual time=10.582..10.582 rows=2 loops=1)
    -> Table scan on <temporary>  (actual time=10.567..10.567 rows=2 loops=1)
        -> Aggregate using temporary table  (actual time=10.565..10.565 rows=2 loops=1)
            -> Nested loop inner join  (cost=3675681.01 rows=11895) (actual time=2.049..10.527 rows=6 loops=1)
                -> Filter: ((wh.UserId = 123) and (wh.`Date` between <cache>(('2024-10-25' - interval 30 day)) and '2024-10-25'))  (cost=1.59 rows=3) (actual time=0.040..0.088 rows=6
loops=1)
                    -> Intersect rows sorted by row ID  (cost=1.59 rows=3) (actual time=0.035..0.074 rows=6 loops=1)
                        -> Index range scan on wh using UserId over (UserId = 123 AND '2024-09-25' <= Date <= '2024-10-25')  (cost=0.25 rows=6) (actual time=0.025..0.041 rows=6 loops=
1)
                -> Index lookup on c using Category_idx (Category='Topwear'), with index condition: ((c.ClothId = wh.Cloth1) or (c.ClothId = wh.Cloth2) or (c.ClothId = wh.Cloth3) or (
c.ClothId = wh.Cloth4) or (c.ClothId = wh.Cloth5))  (cost=120.79 rows=3965) (actual time=1.738..1.739 rows=1 loops=6)
 |
```

Secondly, we tried indexing c.Category. Compared to the baseline results, although the cost for a table lookup for c.Category has decreased from 477.01 to 120.79, the overall cost has not decreased but instead increased dramatically. When compared to the baseline result (cost=6066.63), the cost has increased sharply. Therefore, the result shows that adding the Category_idx significantly worsened performance. We believe the reasons for this outcome could be: (1) Low Selectivity with High Cardinality: If some categories have many records, using the index may result in more I/O than a full table scan. (2) Random I/O: The index could cause excessive random disk access, leading to worse performance than sequential full-table scans.

Option 3 Color_idx, Category_idx

```
create index Color_idx on Clothes(Color);
explain analyze SELECT c.Color, COUNT(*) AS ColorFrequency
FROM WearingHistory wh
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR
wh.Cloth3 = c.ClothId
               OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
WHERE wh.UserId = 123 AND c.Category = 'Topwear'
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 30 DAY) AND
```

```
'2024-10-25'
  AND c.Color IS NOT NULL
GROUP BY c.Color
ORDER BY ColorFrequency DESC;
```

```
| -> Sort: ColorFrequency DESC  (actual time=10.269..10.269 rows=2 loops=1)
    -> Table scan on <temporary>  (actual time=10.253..10.253 rows=2 loops=1)
        -> Aggregate using temporary table  (actual time=10.250..10.250 rows=2 loops=1)
            -> Nested loop inner join  (cost=3675681.01 rows=11895) (actual time=1.882..10.213 rows=6 loops=1)
                -> Filter: ((wh.UserId = 123) and (wh.`Date` between <cache>(('2024-10-25' - interval 30 day)) and '2024-10-25'))  (cost=1.59 rows=3) (actual time=0.041..0.093 rows=6
loops=1)
                    -> Intersect rows sorted by row ID  (cost=1.59 rows=3) (actual time=0.035..0.078 rows=6 loops=1)
                        -> Index range scan on wh using UserId over (UserId = 123 AND '2024-09-25' <= Date <= '2024-10-25')  (cost=0.25 rows=6) (actual time=0.026..0.044 rows=6 loops=
1)
                -> Index lookup on c using Category_idx (Category='Topwear'), with index condition: ((c.ClothId = wh.Cloth1) or (c.ClothId = wh.Cloth2) or (c.ClothId = wh.Cloth3) or (
c.ClothId = wh.Cloth4) or (c.ClothId = wh.Cloth5))  (cost=120.79 rows=3965) (actual time=1.685..1.686 rows=1 loops=6)
 |
```

Lastly, We tried indexing c.Color and c.Category together. From the results, we can see that indexing c.Color and c.Category together yields the same outcome as option2. The cost for a table lookup for c.Category remains at 120.79, and the overall cost is still 367581.01. This indicates that combining both attributes to do indexing does not provide any performance improvement. We believe the reasons for this result are related to the color_idx ineffectiveness in option1 and the poor performance with Category_idx in option2.

Conclusion: Option0: baseline is the best

3. List items that are not black, not for sports, and in one of the FavoriteGroups that were worn in the latest 2 weeks

```
SELECT c.ClothName, fg.GroupName, c.Color, c.Usages
FROM Clothes c
JOIN Include i ON c.ClothId = i.ClothId
JOIN FavoriteGroups fg ON i.FavoriteId = fg.FavoriteId
JOIN WearingHistory wh ON (c.ClothId = wh.Cloth1 OR c.ClothId = wh.Cloth2
                           OR c.ClothId = wh.Cloth3 OR c.ClothId =
wh.Cloth4
                           OR c.ClothId = wh.Cloth5)
WHERE c.Color != 'black'
  AND c.Usages != 'sports'
  AND fg.UserId = 2
  AND wh.UserId = 2
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND
'2024-10-25'
GROUP BY fg.GroupName, c.ClothName, c.Color, c.Usages;
```

Option 0 Baseline

```
explain analyze SELECT c.ClothName, fg.GroupName, c.Color, c.Usages
FROM Clothes c
```

```
JOIN Include i ON c.ClothId = i.ClothId
JOIN FavoriteGroups fg ON i.FavoriteId = fg.FavoriteId
JOIN WearingHistory wh ON (c.ClothId = wh.Cloth1 OR c.ClothId = wh.Cloth2
                           OR c.ClothId = wh.Cloth3 OR c.ClothId =
wh.Cloth4
                           OR c.ClothId = wh.Cloth5)
WHERE c.Color != 'black'
  AND c.Usages != 'sports'
  AND fg.UserId = 2
  AND wh.UserId = 2
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND
'2024-10-25'
GROUP BY fg.GroupName, c.ClothName, c.Color, c.Usages;
```

```
-----------------------------------------------------------------+
| -> Table scan on <temporary>  (cost=22.61..22.61 rows=0.4) (actual time=0.413..0.414 rows=5 loops=1)
    -> Temporary table with deduplication  (cost=20.11..20.11 rows=0.4) (actual time=0.411..0.411 rows=5 loops=1)
        -> Nested loop inner join  (cost=20.07 rows=0.4) (actual time=0.184..0.369 rows=6 loops=1)
            -> Nested loop inner join  (cost=19.99 rows=0.5) (actual time=0.163..0.326 rows=8 loops=1)
                -> Nested loop inner join  (cost=9.53 rows=9) (actual time=0.145..0.222 rows=18 loops=1)
                    -> Filter: ((wh.UserId = 2) and (wh.`Date` between <cache>(('2024-10-25' - interval 14 day)) and '2024-10-25'))  (cost=3.00 rows=3) (actual
 time=0.123..0.160 rows=6 loops=1)
                        -> Intersect rows sorted by row ID  (cost=3.00 rows=3) (actual time=0.116..0.149 rows=6 loops=1)
                            -> Index range scan on wh using UserId over (UserId = 2 AND '2024-10-11' <= Date <= '2024-10-25')  (cost=0.86 rows=6) (actual time=
0.105..0.113 rows=6 loops=1)
                    -> Index lookup on fg using UserId (UserId=2)  (cost=1.97 rows=3) (actual time=0.009..0.010 rows=3 loops=6)
                -> Filter: ((i.ClothId = wh.Cloth1) or (i.ClothId = wh.Cloth2) or (i.ClothId = wh.Cloth3) or (i.ClothId = wh.Cloth4) or (i.ClothId = wh.Cloth5)
)  (cost=0.80 rows=0.05) (actual time=0.005..0.006 rows=0 loops=18)
                    -> Covering index lookup on i using PRIMARY (FavoriteId=fg.FavoriteId)  (cost=0.80 rows=4) (actual time=0.002..0.003 rows=3 loops=18)
            -> Filter: ((c.Color <> 'black') and (c.Usages <> 'sports'))  (cost=0.26 rows=1) (actual time=0.005..0.005 rows=1 loops=8)
                -> Single-row index lookup on c using PRIMARY (ClothId=i.ClothId)  (cost=0.26 rows=1) (actual time=0.003..0.003 rows=1 loops=8)
 |
```

Without any indexing, the baseline query has a cost for table scan on <temporary> of 22.61. For further optimization, we chose to index Clothes(Usages) and Clothes(Color) because these attributes are crucial for filtering in the query and do not serve as primary or foreign keys.

Option1 Adding index Color_idx on Clothes(Color);

```
create index Color_idx on Clothes(Color);
explain analyze SELECT c.ClothName, fg.GroupName, c.Color, c.Usages
FROM Clothes c
JOIN Include i ON c.ClothId = i.ClothId
JOIN FavoriteGroups fg ON i.FavoriteId = fg.FavoriteId
JOIN WearingHistory wh ON (c.ClothId = wh.Cloth1 OR c.ClothId = wh.Cloth2
                           OR c.ClothId = wh.Cloth3 OR c.ClothId =
wh.Cloth4
                           OR c.ClothId = wh.Cloth5)
WHERE c.Color != 'black'
  AND c.Usages != 'sports'
  AND fg.UserId = 2
  AND wh.UserId = 2
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND
'2024-10-25'
```

```
GROUP BY fg.GroupName, c.ClothName, c.Color, c.Usages;
```

```
-----------------------------------------------------------------+
| -> Table scan on <temporary>  (cost=22.59..22.59 rows=0.2) (actual time=0.288..0.289 rows=5 loops=1)
    -> Temporary table with deduplication  (cost=20.09..20.09 rows=0.2) (actual time=0.286..0.286 rows=5 loops=1)
        -> Nested loop inner join  (cost=20.07 rows=0.2) (actual time=0.091..0.254 rows=6 loops=1)
            -> Nested loop inner join  (cost=19.99 rows=0.5) (actual time=0.081..0.226 rows=8 loops=1)
                -> Nested loop inner join  (cost=9.53 rows=9) (actual time=0.066..0.160 rows=18 loops=1)
                    -> Filter: ((wh.UserId = 2) and (wh.`Date` between <cache>(('2024-10-25' - interval 14 day)) and '2024-10-25'))  (cost=3.00 rows=3) (actual
 time=0.047..0.103 rows=6 loops=1)
                        -> Intersect rows sorted by row ID  (cost=3.00 rows=3) (actual time=0.040..0.093 rows=6 loops=1)
                            -> Index range scan on wh using UserId over (UserId = 2 AND '2024-10-11' <= Date <= '2024-10-25')  (cost=0.86 rows=6) (actual time=
0.031..0.061 rows=6 loops=1)
                    -> Index lookup on fg using UserId (UserId=2)  (cost=1.97 rows=3) (actual time=0.008..0.009 rows=3 loops=6)
                -> Filter: ((i.ClothId = wh.Cloth1) or (i.ClothId = wh.Cloth2) or (i.ClothId = wh.Cloth3) or (i.ClothId = wh.Cloth4) or (i.ClothId = wh.Cloth5)
)  (cost=0.80 rows=0.05) (actual time=0.003..0.003 rows=0 loops=18)
                    -> Covering index lookup on i using PRIMARY (FavoriteId=fg.FavoriteId)  (cost=0.80 rows=4) (actual time=0.002..0.003 rows=3 loops=18)
            -> Filter: ((c.Color <> 'black') and (c.Usages <> 'sports'))  (cost=0.19 rows=0.5) (actual time=0.003..0.003 rows=1 loops=8)
                -> Single-row index lookup on c using PRIMARY (ClothId=i.ClothId)  (cost=0.19 rows=1) (actual time=0.002..0.002 rows=1 loops=8)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

First, we added an index on Color, which reduced the cost for table scan on <temporary> slightly from 22.61 to 22.59. With Color_idx, we could filter black items more efficiently, reducing the cost of filtering on Color from 0.26 to 0.19. However, the overall improvement remains minimal, as the majority of the query cost comes from other operations. Additionally, since the Color column contains many repeated values, the index provides limited benefit. In this situation, scanning the index can become almost as costly as performing a full table scan.

Option2 Adding index Usages_idx on Clothes(Usages) & Color_idx on Clothes(Color);

```
create index Usages_idx on Clothes(Usages);
explain analyze SELECT c.ClothName, fg.GroupName, c.Color, c.Usages
FROM Clothes c
JOIN Include i ON c.ClothId = i.ClothId
JOIN FavoriteGroups fg ON i.FavoriteId = fg.FavoriteId
JOIN WearingHistory wh ON (c.ClothId = wh.Cloth1 OR c.ClothId = wh.Cloth2
                           OR c.ClothId = wh.Cloth3 OR c.ClothId =
wh.Cloth4
                           OR c.ClothId = wh.Cloth5)
WHERE c.Color != 'black'
  AND c.Usages != 'sports'
  AND fg.UserId = 2
  AND wh.UserId = 2
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND
'2024-10-25'
GROUP BY fg.GroupName, c.ClothName, c.Color, c.Usages;
```

```
-----------------------------------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (cost=19.43..20.20 rows=1) (actual time=0.668..0.669 rows=5 loops=1)
    -> Temporary table with deduplication  (cost=17.68..17.68 rows=1) (actual time=0.664..0.664 rows=5 loops=1)
        -> Filter: ((wh.Cloth1 = i.ClothId) or (wh.Cloth2 = i.ClothId) or (wh.Cloth3 = i.ClothId) or (wh.Cloth4 = i.ClothId) or (wh.Cloth5 = i.ClothId))  (cost
=17.54 rows=1) (actual time=0.517..0.619 rows=6 loops=1)
            -> Inner hash join (no condition)  (cost=17.54 rows=1) (actual time=0.513..0.572 rows=54 loops=1)
                -> Filter: ((wh.UserId = 2) and (wh.`Date` between <cache>(('2024-10-25' - interval 14 day)) and '2024-10-25'))  (cost=1.12 rows=1) (actual tim
e=0.199..0.246 rows=6 loops=1)
                    -> Intersect rows sorted by row ID  (cost=1.12 rows=3) (actual time=0.187..0.229 rows=6 loops=1)
                        -> Index range scan on wh using UserId over (UserId = 2 AND '2024-10-11' <= Date <= '2024-10-25')  (cost=0.86 rows=6) (actual time=0.17
1..0.184 rows=6 loops=1)
                -> Hash
                    -> Nested loop inner join  (cost=9.47 rows=3) (actual time=0.213..0.261 rows=9 loops=1)
                        -> Nested loop inner join  (cost=5.66 rows=11) (actual time=0.195..0.210 rows=10 loops=1)
                            -> Index lookup on fg using UserId (UserId=2)  (cost=2.17 rows=3) (actual time=0.180..0.181 rows=3 loops=1)
                            -> Covering index lookup on i using PRIMARY (FavoriteId=fg.FavoriteId)  (cost=0.92 rows=4) (actual time=0.007..0.009 rows=3 loops=3
)
                        -> Filter: ((c.Color <> 'black') and (c.Usages <> 'sports'))  (cost=0.25 rows=0.3) (actual time=0.005..0.005 rows=1 loops=10)
                            -> Single-row index lookup on c using PRIMARY (ClothId=i.ClothId)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=10)
 |
```

By adding indexes on both Usages and Color, we reduced the cost for table scan on <temporary> 22.61 in the baseline to 19.43. Although the combined filtering cost on Color and Usages slightly increases to 0.25 (compared to 0.19 when filtering only on Color in Option 1), the overall query cost is significantly lower. This improvement may be due to the introduction of a hash join and an index range scan on the WearingHistory table, based on the UserId and date range conditions, with a cost of 1.12. Together, these optimizations contribute to the overall cost reduction, resulting in the best performance for Option 2.

Option3 Adding index Usages_idx on Clothes(Usages);

```sql
Drop index Color_idx on Clothes;
explain analyze SELECT c.ClothName, fg.GroupName, c.Color, c.Usages
FROM Clothes c
JOIN Include i ON c.ClothId = i.ClothId
JOIN FavoriteGroups fg ON i.FavoriteId = fg.FavoriteId
JOIN WearingHistory wh ON (c.ClothId = wh.Cloth1 OR c.ClothId = wh.Cloth2
                           OR c.ClothId = wh.Cloth3 OR c.ClothId =
wh.Cloth4
                           OR c.ClothId = wh.Cloth5)
WHERE c.Color != 'black'
  AND c.Usages != 'sports'
  AND fg.UserId = 2
  AND wh.UserId = 2
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND
'2024-10-25'
GROUP BY fg.GroupName, c.ClothName, c.Color, c.Usages;
```

```
-----------------------------------------------------------------------+
| -> Table scan on <temporary>  (cost=22.59..22.59 rows=0.2) (actual time=0.485..0.486 rows=5 loops=1)
    -> Temporary table with deduplication  (cost=20.09..20.09 rows=0.2) (actual time=0.483..0.483 rows=5 loops=1)
        -> Nested loop inner join  (cost=20.07 rows=0.2) (actual time=0.191..0.430 rows=6 loops=1)
            -> Nested loop inner join  (cost=19.99 rows=0.5) (actual time=0.174..0.386 rows=8 loops=1)
                -> Nested loop inner join  (cost=9.53 rows=9) (actual time=0.128..0.262 rows=18 loops=1)
                    -> Filter: ((wh.UserId = 2) and (wh.`Date` between <cache>(('2024-10-25' - interval 14 day)) and '2024-10-25'))  (cost=3.00 rows=3) (actual
 time=0.099..0.149 rows=6 loops=1)
                        -> Intersect rows sorted by row ID  (cost=3.00 rows=3) (actual time=0.085..0.130 rows=6 loops=1)
                            -> Index range scan on wh using UserId over (UserId = 2 AND '2024-10-11' <= Date <= '2024-10-25')  (cost=0.86 rows=6) (actual time=
0.063..0.075 rows=6 loops=1)
                    -> Index lookup on fg using UserId (UserId=2)  (cost=1.97 rows=3) (actual time=0.017..0.018 rows=3 loops=6)
                -> Filter: ((i.ClothId = wh.Cloth1) or (i.ClothId = wh.Cloth2) or (i.ClothId = wh.Cloth3) or (i.ClothId = wh.Cloth4) or (i.ClothId = wh.Cloth5)
)  (cost=0.80 rows=0.05) (actual time=0.006..0.007 rows=0 loops=18)
                    -> Covering index lookup on i using PRIMARY (FavoriteId=fg.FavoriteId)  (cost=0.80 rows=4) (actual time=0.004..0.005 rows=3 loops=18)
            -> Filter: ((c.Color <> 'black') and (c.Usages <> 'sports'))  (cost=0.18 rows=0.5) (actual time=0.005..0.005 rows=1 loops=8)
                -> Single-row index lookup on c using PRIMARY (ClothId=i.ClothId)  (cost=0.18 rows=1) (actual time=0.004..0.004 rows=1 loops=8)
 |
+-----------------------------------------------------------------------------------------------------------------------------------+
```

By only adding an index on Usages, the cost for table scan on <temporary> is 22.59, exactly the same as in Option 1, where we only indexed Color. Similar to Option 1, indexing a single attribute provides minimal improvement over the baseline. The limited impact likely results from the query's performance bottleneck, which may lie in the complex joins. Additionally, the Usages column contains many repeated values, reducing the effectiveness of the index.

Conclusion: Option 2: Adding Usages_idx & Color_idx is the best

4. **Query: List ClothId of some ''Category" that were often worn (time > threshold within the last two weeks) for certain 'Usages' but not in any favorite group**

Option0: baseline

```
explain analyze (
SELECT ClothId
FROM WearingHistory wh
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR wh.Cloth3 = c.ClothId
               OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
WHERE wh.UserId = 120 AND c.Usages = 'Casual' AND c.Category = 'Topwear'
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND '2024-10-25'
  AND c.ClothId IS NOT NULL
GROUP BY c.ClothId
HAVING COUNT(*) >= 3
) EXCEPT (
SELECT ClothId
FROM Include NATURAL JOIN FavoriteGroups NATURAL JOIN Clothes
WHERE UserId = 120
);
```

```
| -> Table scan on <except temporary>  (cost=5.35..5.35 rows=0) (actual time=14.704..14.704 rows=2 loops=1)
    -> Except materialize with deduplication  (cost=2.85..2.85 rows=0) (actual time=14.703..14.703 rows=2 loops=1)
        -> Filter: (count(0) >= 3)  (actual time=14.662..14.663 rows=2 loops=1)
            -> Table scan on <temporary>  (actual time=14.657..14.658 rows=2 loops=1)
                -> Aggregate using temporary table  (actual time=14.654..14.654 rows=2 loops=1)
                    -> Filter: ((c.ClothId = wh.Cloth1) or (c.ClothId = wh.Cloth2) or (c.ClothId = wh.Cloth3) or (c.ClothId = wh.Cloth4) or (c.ClothId = wh.Cloth5))  (cost=1573.31 row
s=0.0002) (actual time=1.823..14.633 rows=6 loops=1)
                        -> Inner hash join (no condition)  (cost=1573.31 rows=0.0002) (actual time=0.165..10.679 rows=20334 loops=1)
                            -> Filter: ((c.Category = 'Topwear') and (c.Usages = 'Casual'))  (cost=523.37 rows=1) (actual time=0.030..8.667 rows=3389 loops=1)
                                -> Table scan on c  (cost=523.37 rows=15447) (actual time=0.025..5.832 rows=15911 loops=1)
                            -> Hash
                                -> Filter: ((wh.UserId = 120) and (wh.`Date` between <cache>(('2024-10-25' - interval 14 day)) and '2024-10-25'))  (cost=3.00 rows=3) (actual time=0.08
0..0.104 rows=6 loops=1)
                                    -> Intersect rows sorted by row ID  (cost=3.00 rows=3) (actual time=0.073..0.094 rows=6 loops=1)
                                        -> Index range scan on wh using UserId over (UserId = 120 AND '2024-10-11' <= Date <= '2024-10-25')  (cost=0.86 rows=6) (actual time=0.062..0.0
70 rows=6 loops=1)
        -> Nested loop inner join  (cost=2.85 rows=0.2) (actual time=0.022..0.022 rows=0 loops=1)
            -> Nested loop inner join  (cost=1.58 rows=4) (actual time=0.022..0.022 rows=0 loops=1)
                -> Covering index lookup on FavoriteGroups using UserId (UserId=120)  (cost=0.35 rows=1) (actual time=0.021..0.021 rows=0 loops=1)
                -> Covering index lookup on Include using PRIMARY (FavoriteId=FavoriteGroups.FavoriteId)  (cost=1.23 rows=4) (never executed)
            -> Filter: (Clothes.UserId = 120)  (cost=0.25 rows=0.05) (never executed)
                -> Single-row index lookup on Clothes using PRIMARY (ClothId=Include.ClothId)  (cost=0.25 rows=1) (never executed)
```

Option1: Adding index usages_idx on Clothes(Usages)

```
Create index usages_idx on Clothes(Usages);

explain analyze (
SELECT ClothId
FROM WearingHistory wh
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR wh.Cloth3 = c.ClothId
               OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
WHERE wh.UserId = 120 AND c.Usages = 'Casual' AND c.Category = 'Topwear'
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND '2024-10-25'
  AND c.ClothId IS NOT NULL
```

```
GROUP BY c.ClothId
HAVING COUNT(*) >= 3
) EXCEPT (
SELECT ClothId
FROM Include NATURAL JOIN FavoriteGroups NATURAL JOIN Clothes
WHERE UserId = 120
);

Drop index usages_idx on Clothes;
```

```
| -> Table scan on <except temporary>  (cost=5.35..5.35 rows=0) (actual time=14.735..14.736 rows=2 loops=1)
    -> Except materialize with deduplication  (cost=2.85..2.85 rows=0) (actual time=14.734..14.734 rows=2 loops=1)
        -> Filter: (count(0) >= 3)  (actual time=14.698..14.698 rows=2 loops=1)
            -> Table scan on <temporary>  (actual time=14.693..14.693 rows=2 loops=1)
                -> Aggregate using temporary table  (actual time=14.690..14.690 rows=2 loops=1)
                    -> Filter: ((c.ClothId = wh.Cloth1) or (c.ClothId = wh.Cloth2) or (c.ClothId = wh.Cloth3) or (c.ClothId = wh.Cloth4) or (c.ClothId = wh.Cloth5))  (cost=1511.50 row
s=0.0002) (actual time=2.730..14.668 rows=6 loops=1)
                        -> Inner hash join (no condition)  (cost=1511.50 rows=0.0002) (actual time=0.610..10.816 rows=20334 loops=1)
                            -> Filter: ((c.Category = 'Topwear') and (c.Usages = 'Casual'))  (cost=502.77 rows=1) (actual time=0.523..8.887 rows=3389 loops=1)
                                -> Table scan on c  (cost=502.77 rows=15447) (actual time=0.518..6.154 rows=15911 loops=1)
                            -> Hash
                                -> Filter: ((wh.UserId = 120) and (wh.`Date` between <cache>(('2024-10-25' - interval 14 day)) and '2024-10-25'))  (cost=3.00 rows=3) (actual time=0.05
1..0.075 rows=6 loops=1)
                                    -> Intersect rows sorted by row ID  (cost=3.00 rows=3) (actual time=0.043..0.065 rows=6 loops=1)
                                        -> Index range scan on wh using UserId over (UserId = 120 AND '2024-10-11' <= Date <= '2024-10-25')  (cost=0.86 rows=6) (actual time=0.026..0.0
33 rows=6 loops=1)
        -> Nested loop inner join  (cost=2.85 rows=0.2) (actual time=0.020..0.020 rows=0 loops=1)
            -> Nested loop inner join  (cost=1.58 rows=4) (actual time=0.020..0.020 rows=0 loops=1)
                -> Covering index lookup on FavoriteGroups using UserId (UserId=120)  (cost=0.35 rows=1) (actual time=0.019..0.019 rows=0 loops=1)
                -> Covering index lookup on Include using PRIMARY (FavoriteId=FavoriteGroups.FavoriteId)  (cost=1.23 rows=4) (never executed)
            -> Filter: (Clothes.UserId = 120)  (cost=0.25 rows=0.05) (never executed)
                -> Single-row index lookup on Clothes using PRIMARY (ClothId=Include.ClothId)  (cost=0.25 rows=1) (never executed)
 |
```

The query does not use the index lookup instead of table scan when filtering Usages and Category, but the cost of filtering Usages and Category has dropped from 523.37 to 502.77. The cost of Hash join has also been reduced.

Option2: Adding index category_idx on Clothes(Category)

```
Create index category_idx on Clothes(Category);

explain analyze (
SELECT ClothId
FROM WearingHistory wh
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR wh.Cloth3 = c.ClothId
               OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
WHERE wh.UserId = 120 AND c.Usages = 'Casual' AND c.Category = 'Topwear'
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND '2024-10-25'
  AND c.ClothId IS NOT NULL
GROUP BY c.ClothId
HAVING COUNT(*) >= 3
) EXCEPT (
SELECT ClothId
FROM Include NATURAL JOIN FavoriteGroups NATURAL JOIN Clothes
WHERE UserId = 120
);

Drop index category_idx on Clothes;
```

```
| -> Table scan on <except temporary>  (cost=5.35..5.35 rows=0) (actual time=10.466..10.466 rows=2 loops=1)
    -> Except materialize with deduplication  (cost=2.85..2.85 rows=0) (actual time=10.465..10.465 rows=2 loops=1)
        -> Filter: (count(0) >= 3)  (actual time=10.441..10.442 rows=2 loops=1)
            -> Table scan on <temporary>  (actual time=10.437..10.438 rows=2 loops=1)
                -> Aggregate using temporary table  (actual time=10.435..10.435 rows=2 loops=1)
                    -> Nested loop inner join  (cost=1554.75 rows=1) (actual time=1.890..10.410 rows=6 loops=1)
                        -> Filter: ((wh.UserId = 120) and (wh.`Date` between <cache>(('2024-10-25' - interval 14 day)) and '2024-10-25'))  (cost=3.00 rows=3) (actual time=0.044..0.107
 rows=6 loops=1)
                            -> Intersect rows sorted by row ID  (cost=3.00 rows=3) (actual time=0.037..0.090 rows=6 loops=1)
                                -> Index range scan on wh using UserId over (UserId = 120 AND '2024-10-11' <= Date <= '2024-10-25')  (cost=0.86 rows=6) (actual time=0.029..0.045 rows=
6 loops=1)
                        -> Filter: (c.Usages = 'Casual')  (cost=120.76 rows=0.3) (actual time=1.715..1.717 rows=1 loops=6)
                            -> Index lookup on c using category_idx (Category='Topwear'), with index condition: ((c.ClothId = wh.Cloth1) or (c.ClothId = wh.Cloth2) or (c.ClothId = wh.
Cloth3) or (c.ClothId = wh.Cloth4) or (c.ClothId = wh.Cloth5))  (cost=120.76 rows=3965) (actual time=1.714..1.715 rows=1 loops=6)
        -> Nested loop inner join  (cost=2.85 rows=0.2) (actual time=0.013..0.013 rows=0 loops=1)
            -> Nested loop inner join  (cost=1.58 rows=4) (actual time=0.012..0.012 rows=0 loops=1)
                -> Covering index lookup on FavoriteGroups using UserId (UserId=120)  (cost=0.35 rows=1) (actual time=0.012..0.012 rows=0 loops=1)
                -> Covering index lookup on Include using PRIMARY (FavoriteId=FavoriteGroups.FavoriteId)  (cost=1.23 rows=4) (never executed)
            -> Filter: (Clothes.UserId = 120)  (cost=0.25 rows=0.05) (never executed)
                -> Single-row index lookup on Clothes using PRIMARY (ClothId=Include.ClothId)  (cost=0.25 rows=1) (never executed)
 |
```

By adding category_idx, the database starts using the index to filter Category, which leads to the decrease of cost of filtering Usages and Category from 523.37 for table scan to 120.76 for index lookup. In addition, the cost of nested loop inner join(1554.75) is less than inner hash join(1573.31) of baseline.

Option3: Adding both usages_idx on Clothes(Usages) and category_idx on Clothes(Category)

```
Create index usages_idx on Clothes(Usages);
Create index category_idx on Clothes(Category);

explain analyze (
SELECT ClothId
FROM WearingHistory wh
JOIN Clothes c ON wh.Cloth1 = c.ClothId OR wh.Cloth2 = c.ClothId OR wh.Cloth3 = c.ClothId
               OR wh.Cloth4 = c.ClothId OR wh.Cloth5 = c.ClothId
WHERE wh.UserId = 120 AND c.Usages = 'Casual' AND c.Category = 'Topwear'
  AND wh.Date BETWEEN DATE_SUB('2024-10-25', INTERVAL 14 DAY) AND '2024-10-25'
  AND c.ClothId IS NOT NULL
GROUP BY c.ClothId
HAVING COUNT(*) >= 3
) EXCEPT (
SELECT ClothId
FROM Include NATURAL JOIN FavoriteGroups NATURAL JOIN Clothes
WHERE UserId = 120
);

Drop index usages_idx on Clothes;
Drop index category_idx on Clothes;
```

```
| -> Table scan on <except temporary>  (cost=5.35..5.35 rows=0) (actual time=12.153..12.154 rows=2 loops=1)
    -> Except materialize with deduplication  (cost=2.85..2.85 rows=0) (actual time=12.152..12.152 rows=2 loops=1)
        -> Filter: (count(0) >= 3)  (actual time=12.116..12.117 rows=2 loops=1)
            -> Table scan on <temporary>  (actual time=12.113..12.113 rows=2 loops=1)
                -> Aggregate using temporary table  (actual time=12.110..12.110 rows=2 loops=1)
                    -> Nested loop inner join  (cost=1554.75 rows=2) (actual time=2.759..12.071 rows=6 loops=1)
                        -> Filter: ((wh.UserId = 120) and (wh.`Date` between <cache>(('2024-10-25' - interval 14 day)) and '2024-10-25'))  (cost=3.00 rows=3) (actual time=0.064..0.373
 rows=6 loops=1)
                            -> Intersect rows sorted by row ID  (cost=3.00 rows=3) (actual time=0.055..0.346 rows=6 loops=1)
                                -> Index range scan on wh using UserId over (UserId = 120 AND '2024-10-11' <= Date <= '2024-10-25')  (cost=0.86 rows=6) (actual time=0.039..0.067 rows=
6 loops=1)
                        -> Filter: (c.Usages = 'Casual')  (cost=120.77 rows=1) (actual time=1.947..1.949 rows=1 loops=6)
                            -> Index lookup on c using category_idx (Category='Topwear'), with index condition: ((c.ClothId = wh.Cloth1) or (c.ClothId = wh.Cloth2) or (c.ClothId = wh.
Cloth3) or (c.ClothId = wh.Cloth4) or (c.ClothId = wh.Cloth5))  (cost=120.77 rows=3965) (actual time=1.945..1.947 rows=1 loops=6)
        -> Nested loop inner join  (cost=2.85 rows=0.2) (actual time=0.013..0.013 rows=0 loops=1)
            -> Nested loop inner join  (cost=1.58 rows=4) (actual time=0.013..0.013 rows=0 loops=1)
                -> Covering index lookup on FavoriteGroups using UserId (UserId=120)  (cost=0.35 rows=1) (actual time=0.012..0.012 rows=0 loops=1)
                -> Covering index lookup on Include using PRIMARY (FavoriteId=FavoriteGroups.FavoriteId)  (cost=1.23 rows=4) (never executed)
            -> Filter: (Clothes.UserId = 120)  (cost=0.25 rows=0.05) (never executed)
                -> Single-row index lookup on Clothes using PRIMARY (ClothId=Include.ClothId)  (cost=0.25 rows=1) (never executed)
 |
```

The result of indexing both Usages and Category is similar to the result of indexing Category. The cost of filtering Usages and Category slightly increases from 120.76 to 120.77. We believe

it is because of the overhead of indexing and the query optimizer does not choose to leverage the index on Cloth(Usages).

Conclusion: Option2: Adding index category_idx on Clothes(Category) is the best