ClimaCloset - Weather Based Outfit Suggestion App
Jack Moran, Ben Goldman, Livia Bezati, and Reda Mehdaoui
Stage 3 - Team 085

UserDatabase

Our database includes 5 tables: users, clothingItems, outfits, weather, and restaurants. These tables are shown below along with the DDL commands for each table.

```
mysql> show tables;
+------------------------+
| Tables_in_UserDatabase |
+------------------------+
| clothingItems          |
| outfits                |
| restaurants            |
| users                  |
| weather                |
+------------------------+
5 rows in set (0.00 sec)
```

- **clothingItems**:

```
mysql> select count(*) from clothingItems;
+----------+
| count(*) |
+----------+
|     2000 |
+----------+
1 row in set (0.00 sec)

mysql> describe clothingItems;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| clothingId | int         | NO   | PRI | NULL    |       |
| name       | varchar(50) | NO   |     | NULL    |       |
| type       | varchar(50) | NO   |     | NULL    |       |
| style      | varchar(50) | NO   |     | NULL    |       |
| color      | varchar(50) | NO   |     | NULL    |       |
| userId     | int         | YES  | MUL | NULL    |       |
+------------+-------------+------+-----+---------+-------+
6 rows in set (0.02 sec)
```

DDL Command:

```
CREATE TABLE clothingItems (
    clothingId INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    type VARCHAR(50) NOT NULL,
```

```
    style VARCHAR(50) NOT NULL,
    color VARCHAR(50) NOT NULL,
    userId INT,
    FOREIGN KEY (userId) REFERENCES users(userId)
);
```

- **outfits**:

```
mysql> select count(*) from outfits;
+----------+
| count(*) |
+----------+
|      445 |
+----------+
1 row in set (0.02 sec)

mysql> describe outfits;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| outfitid    | int         | NO   | PRI | NULL    |       |
| topLayer1   | int         | YES  | MUL | NULL    |       |
| topLayer2   | int         | YES  | MUL | NULL    |       |
| bottom      | int         | YES  | MUL | NULL    |       |
| outfitStyle | varchar(50) | NO   |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

DDL Command:

```
CREATE TABLE outfits (
    outfitid INT PRIMARY KEY,
    topLayer1 INT,
    topLayer2 INT,
    bottom INT,
    outfitStyle VARCHAR(50) NOT NULL,
    FOREIGN KEY (topLayer1) REFERENCES clothingItems(clothingId),
    FOREIGN KEY (topLayer2) REFERENCES clothingItems(clothingId),
    FOREIGN KEY (bottom) REFERENCES clothingItems(clothingId)
);
```

- **restaurants**:

```
mysql> select count(*) from restaurants;
+----------+
| count(*) |
+----------+
|    51717 |
+----------+
1 row in set (0.25 sec)

mysql> describe restaurants;
+--------------+---------------+------+-----+---------+-------+
| Field        | Type          | Null | Key | Default | Extra |
+--------------+---------------+------+-----+---------+-------+
| url          | varchar(255)  | YES  |     | NULL    |       |
| address      | varchar(255)  | YES  |     | NULL    |       |
| name         | varchar(50)   | YES  |     | NULL    |       |
| online_order | tinyint(1)    | YES  |     | NULL    |       |
| book_table   | tinyint(1)    | YES  |     | NULL    |       |
| rate         | decimal(10,2) | YES  |     | NULL    |       |
| votes        | int           | YES  | MUL | NULL    |       |
| phone        | int           | YES  |     | NULL    |       |
| location     | varchar(50)   | YES  | MUL | NULL    |       |
| rest_type    | varchar(50)   | YES  |     | NULL    |       |
+--------------+---------------+------+-----+---------+-------+
10 rows in set (0.00 sec)
```

DDL Command:

```
CREATE TABLE restaurants (
    url          VARCHAR(255) ,
    address      VARCHAR(255) ,
    name         VARCHAR(50) ,
    online_order TINYINT(1) ,
    book_table   TINYINT(1) ,
    rate         DECIMAL(10,2) ,
    votes        INT ,
    phone        INT ,
    location     VARCHAR(50) ,
    rest_type    VARCHAR(50) ,
);
```

- **users**:

```
mysql> select count(*) from users;
+----------+
| count(*) |
+----------+
|      100 |
+----------+
1 row in set (0.04 sec)

mysql> describe users;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| userId    | int          | NO   | PRI | NULL    | auto_increment |
| username  | varchar(50)  | NO   | UNI | NULL    |                |
| password  | varchar(255) | NO   |     | NULL    |                |
| firstName | varchar(50)  | NO   |     | NULL    |                |
| lastName  | varchar(50)  | NO   |     | NULL    |                |
| address   | varchar(255) | NO   |     | NULL    |                |
| location  | varchar(100) | NO   |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
7 rows in set (0.02 sec)
```

DDL Command:

```
CREATE TABLE users (
    userId INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    firstName VARCHAR(50) NOT NULL,
    lastName VARCHAR(50) NOT NULL,
    address VARCHAR(255) NOT NULL,
    location VARCHAR(100) NOT NULL
);
```

- **weather**:

```
mysql> select count(*) from weather;
+----------+
| count(*) |
+----------+
|     7320 |
+----------+
1 row in set (0.02 sec)

mysql> describe weather;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| station     | varchar(20)  | YES  |     | NULL    |       |
| date        | varchar(20)  | YES  |     | NULL    |       |
| latitude    | float        | YES  |     | NULL    |       |
| longitude   | float        | YES  |     | NULL    |       |
| elevation   | float        | YES  |     | NULL    |       |
| location    | varchar(100) | YES  |     | NULL    |       |
| month       | int          | YES  |     | NULL    |       |
| day         | int          | YES  |     | NULL    |       |
| hour        | int          | YES  |     | NULL    |       |
| dailyAvg    | float        | YES  |     | NULL    |       |
| dailyAvgMax | float        | YES  |     | NULL    |       |
| dailyAvgMin | float        | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
12 rows in set (0.01 sec)
```

DDL Command:

```
CREATE TABLE weather (
    station VARCHAR(20),
    date VARCHAR(20),
    latitude FLOAT,
    longitude FLOAT,
       elevation FLOAT,
       location VARCHAR(100),
       month INT,
       day INT,
       hour INT,
       dailyAvg FLOAT,
       dailyAvgMax FLOAT,
       dailyAvgMin FLOAT
);
```

QUERY #1

SELECT c.userId, o.outfitStyle, COUNT(*) AS styleCount
FROM outfits o
JOIN clothingItems c ON o.topLayer1 = c.clothingId
GROUP BY c.userId, o.outfitStyle
ORDER BY c.userId, styleCount DESC;

This query finds the number of each style of outfits that a user has. This is a query that will potentially be helpful when we try to show a user's closet as we can share how many outfits they can make for each different styles we have.

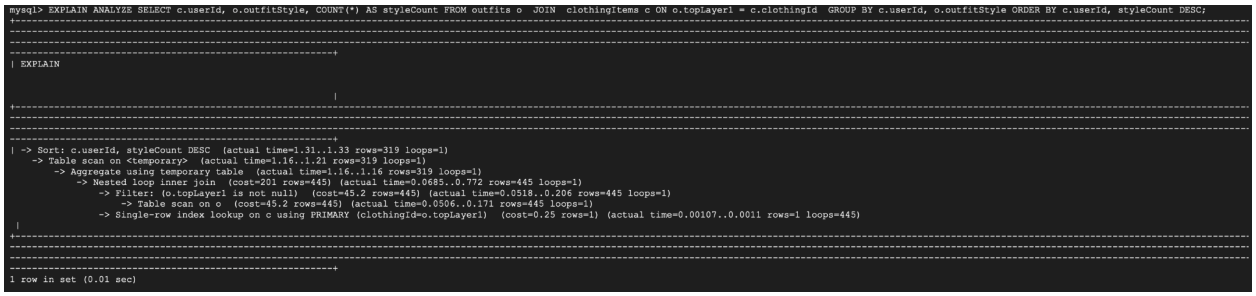
```
mysql> SELECT c.userId, o.outfitStyle, COUNT(*) AS styleCount FROM outfits o JOIN  clothingItems c ON o.topLayer1 = c.clothingId GROUP BY c.userId, o.outfitStyle ORDER BY c.userId, styleCount DESC LIMIT  15;
+--------+----------------+------------+
| userId | outfitStyle    | styleCount |
+--------+----------------+------------+
|      1 | winter         |          2 |
|      1 | summer         |          2 |
|      1 | athletic       |          1 |
|      2 | formal         |          2 |
|      2 | casual         |          1 |
|      3 | athletic       |          3 |
|      3 | formal         |          2 |
|      4 | formal         |          2 |
|      4 | summer         |          1 |
|      4 | athletic       |          1 |
|      5 | winter         |          3 |
|      5 | business casual |          1 |
|      5 | formal         |          1 |
|      6 | winter         |          2 |
|      6 | summer         |          1 |
+--------+----------------+------------+
15 rows in set (0.22 sec)
```

```
mysql> EXPLAIN ANALYZE SELECT c.userId, o.outfitStyle, COUNT(*) AS styleCount
    -> FROM outfits o
    -> JOIN  clothingItems c ON o.topLayer1 = c.clothingId
    -> GROUP BY c.userId, o.outfitStyle
    -> ORDER BY c.userId, styleCount DESC;
+------------------------------------------+
| EXPLAIN                                  |
+------------------------------------------+
| -> Sort: c.userId, styleCount DESC  (actual time=69.7..69.7 rows=319 loops=1)
    -> Table scan on <temporary>  (actual time=68.5..68.6 rows=319 loops=1)
        -> Aggregate using temporary table  (actual time=68.5..68.5 rows=319 loops=1)
            -> Nested loop inner join  (cost=499 rows=445) (actual time=49.2..67.1 rows=445 loops=1)
                -> Filter: (o.topLayer1 is not null)  (cost=46.4 rows=445) (actual time=41.2..44 rows=445 loops=1)
                    -> Table scan on o  (cost=46.4 rows=445) (actual time=41.1..43.9 rows=445 loops=1)
                -> Single-row index lookup on c using PRIMARY (clothingId=o.topLayer1)  (cost=0.917 rows=1) (actual time=0.0518..0.0519 rows=1 loops=445)
 |
+------------------------------------------+
1 row in set (0.11 sec)
```

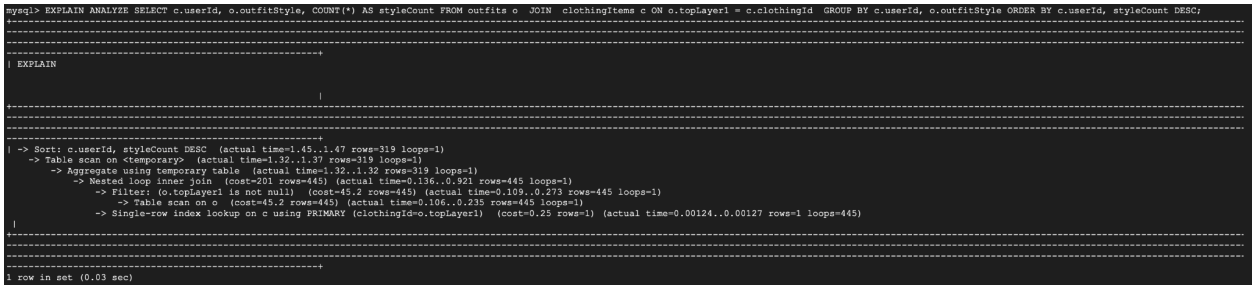
Index #1:
CREATE INDEX idx_toplayer1 ON outfits(topLayer1);

```
mysql> EXPLAIN ANALYZE SELECT c.userId, o.outfitStyle, COUNT(*) AS styleCount FROM outfits o  JOIN  clothingItems c ON o.topLayer1 = c.clothingId  GROUP BY c.userId, o.outfitStyle ORDER BY c.userId, styleCount DESC;
+------------------------------------------+
| EXPLAIN                                  |
+------------------------------------------+
| -> Sort: c.userId, styleCount DESC  (actual time=1.4..1.41 rows=319 loops=1)
    -> Table scan on <temporary>  (actual time=1.25..1.29 rows=319 loops=1)
        -> Aggregate using temporary table  (actual time=1.24..1.24 rows=319 loops=1)
            -> Nested loop inner join  (cost=201 rows=445) (actual time=0.143..0.876 rows=445 loops=1)
                -> Filter: (o.topLayer1 is not null)  (cost=45.2 rows=445) (actual time=0.123..0.277 rows=445 loops=1)
                    -> Table scan on o  (cost=45.2 rows=445) (actual time=0.122..0.245 rows=445 loops=1)
                -> Single-row index lookup on c using PRIMARY (clothingId=o.topLayer1)  (cost=0.25 rows=1) (actual time=0.00108..0.00111 rows=1 loops=445)
 |
+------------------------------------------+
1 row in set (0.01 sec)
```

Index #2:
DROP INDEX idx_toplayer1 ON outfits;
CREATE INDEX idx_userid ON clothingItems(userId);



Index #3:
DROP INDEX idx_userid ON clothingItems;
CREATE INDEX idx_id_user ON clothingItems(clothingId, userId);



	Cost of intended change before
Default	499
idx_toplayer1	201
idx_toplayer1	201
idx_userid	201

We put indexes on the outfits and clothing items tables(join clause and group by). We found a decrease in all of the indexes by the same amount. We ended up deciding that our idx_userid index was the best fitting for this.

QUERY #2

```
SELECT c.userId, o.outfitStyle, COUNT(*) AS styleCount
FROM outfits o
JOIN clothingItems c ON o.topLayer1 = c.clothingId
GROUP BY c.userId, o.outfitStyle
HAVING COUNT(*) = (
   SELECT MAX(style_count)
```

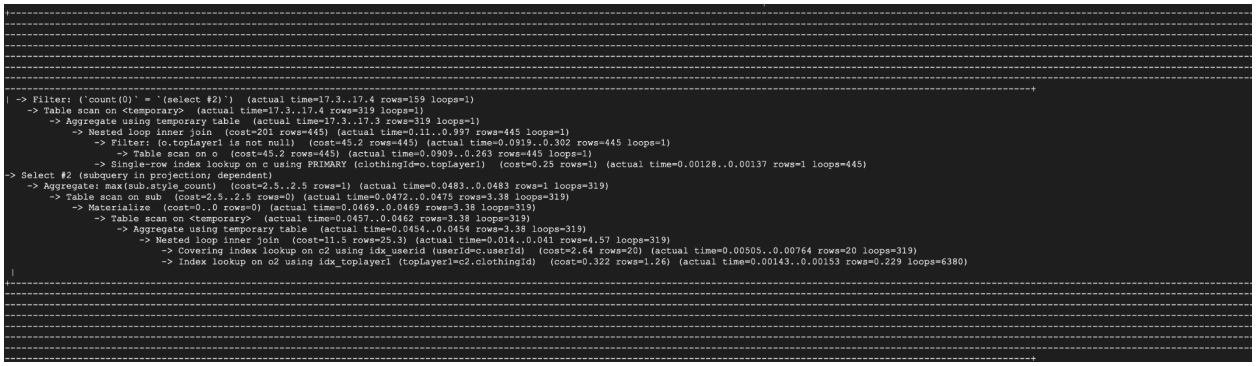
```
    FROM (
        SELECT COUNT(*) AS style_count
        FROM outfits o2
        JOIN clothingItems c2 ON o2.topLayer1 = c2.clothingId
        WHERE c2.userId = c.userId
        GROUP BY o2.outfitStyle
    ) sub
);
```

This query returns the most common outfit styles that a user has. This will be helpful to determine what style may be the user's favorite within their closet, and can help when recommending outfits.

```
mysql> SELECT c.userId, o.outfitStyle, COUNT(*) AS styleCount FROM outfits o JOIN clothingItems c ON o.topLayer1 = c.clothingId GROUP BY c.userId, o.outfitStyle HAVING COUNT(*) = (     SELECT MAX(style_count)     FROM (
     SELECT COUNT(*) AS style_count         FROM outfits o2         JOIN clothingItems c2 ON o2.topLayer1 = c2.clothingId         WHERE c2.userId = c.userId         GROUP BY o2.outfitStyle     ) sub ) LIMIT 15;
+--------+-------------+------------+
| userId | outfitStyle | styleCount |
+--------+-------------+------------+
|      1 | winter      |          2 |
|      1 | summer      |          2 |
|      2 | formal      |          2 |
|      3 | athletic    |          3 |
|      4 | formal      |          2 |
|      5 | winter      |          3 |
|      6 | winter      |          2 |
|      7 | casual      |          3 |
|      8 | winter      |          2 |
|      9 | casual      |          2 |
|     10 | formal      |          1 |
|     10 | winter      |          1 |
|     10 | casual      |          1 |
|     11 | athletic    |          2 |
|     12 | summer      |          2 |
+--------+-------------+------------+
15 rows in set (0.01 sec)
```

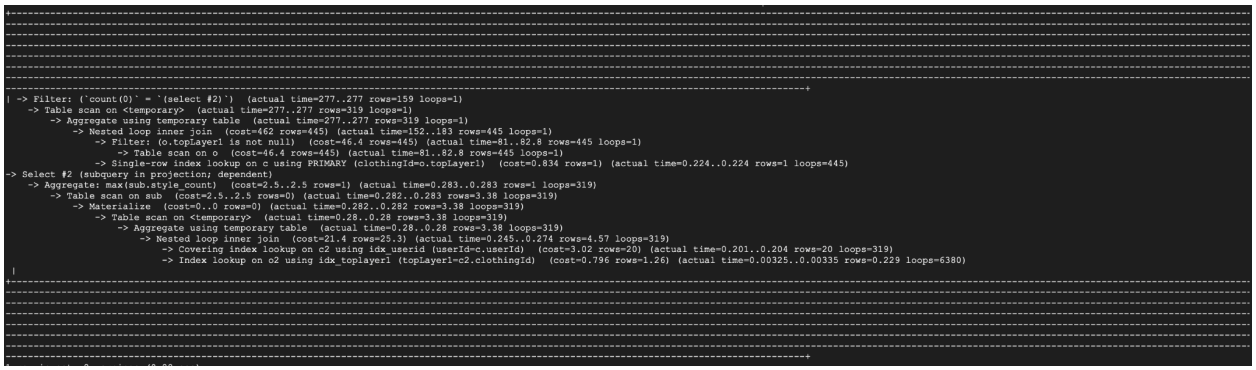
```
mysql> EXPLAIN ANALYZE SELECT c.userId, o.outfitStyle, COUNT(*) AS styleCount FROM outfits o JOIN clothingItems c ON o.topLayer1 = c.clothingId GROUP BY c.userId, o.outfitStyle HAVING COUNT(*) = (     SELECT MAX(style_count
)     FROM (         SELECT COUNT(*) AS style_count         FROM outfits o2         JOIN clothingItems c2 ON o2.topLayer1 = c2.clothingId         WHERE c2.userId = c.userId         GROUP BY o2.outfitStyle     ) sub );
+----------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN
                                                                                                 |
+----------------------------------------------------------------------------------------------------------------------------+
| -> Filter: (`count(0)` = `(select #2)`)  (actual time=22.8..22.9 rows=159 loops=1)
    -> Table scan on <temporary>  (actual time=22.8..22.8 rows=319 loops=1)
        -> Aggregate using temporary table  (actual time=22.8..22.8 rows=319 loops=1)
            -> Nested loop inner join  (cost=201 rows=445) (actual time=0.193..1.5 rows=445 loops=1)
                -> Filter: (o.topLayer1 is not null)  (cost=45.2 rows=445) (actual time=0.137..0.417 rows=445 loops=1)
                    -> Table scan on o  (cost=45.2 rows=445) (actual time=0.133..0.362 rows=445 loops=1)
                -> Single-row index lookup on c using PRIMARY (clothingId=o.topLayer1)  (cost=0.25 rows=1) (actual time=0.0022..0.00223 rows=1 loops=445)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: max(sub.style_count)  (cost=2.5..2.5 rows=1) (actual time=0.0611..0.0611 rows=1 loops=319)
        -> Table scan on sub  (cost=2.5..2.5 rows=0) (actual time=0.0598..0.0602 rows=3.38 loops=319)
            -> Materialize  (cost=0..0 rows=0) (actual time=0.0596..0.0596 rows=3.38 loops=319)
                -> Table scan on <temporary>  (actual time=0.0578..0.0582 rows=3.38 loops=319)
                    -> Aggregate using temporary table  (actual time=0.0575..0.0575 rows=3.38 loops=319)
                        -> Nested loop inner join  (cost=11.5 rows=25.3) (actual time=0.0243..0.0517 rows=4.57 loops=319)
                            -> Covering index lookup on c2 using idx_userid (userId=c.userId)  (cost=2.64 rows=20) (actual time=0.0108..0.0133 rows=20 loops=319)
                            -> Index lookup on o2 using idx_toplayer1 (topLayer1=c2.clothingId)  (cost=0.322 rows=1.26) (actual time=0.00168..0.00178 rows=0.229 loops=6380)
 |
+----------------------------------------------------------------------------------------------------------------------------+
1 row in set, 2 warnings (0.06 sec)
```

Index #1:
CREATE INDEX idx_toplayer1 ON outfits(topLayer1);
DROP INDEX idx_toplayer1 ON outfits;



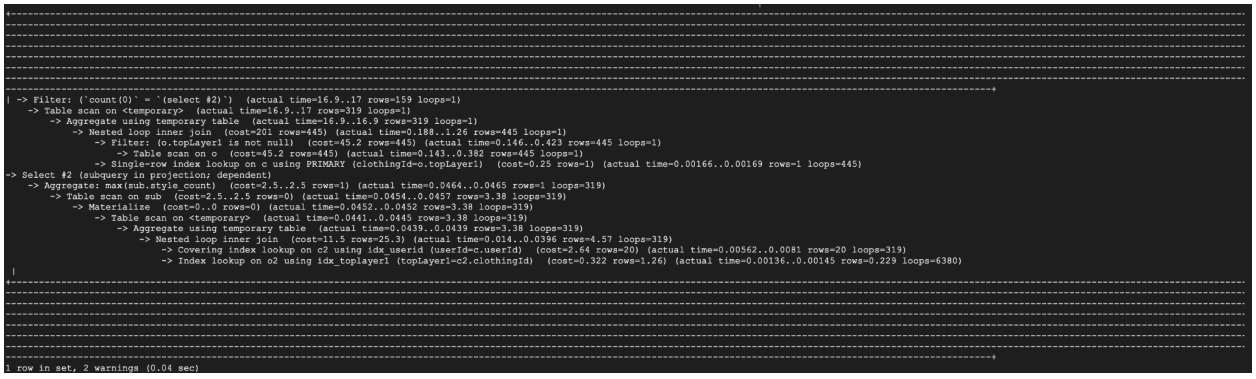
```
| -> Filter: (`count(0)` = `(select #2)`)  (actual time=17.3..17.4 rows=159 loops=1)
    -> Table scan on <temporary>  (actual time=17.3..17.4 rows=319 loops=1)
        -> Aggregate using temporary table  (actual time=17.3..17.3 rows=319 loops=1)
            -> Nested loop inner join  (cost=201 rows=445) (actual time=0.11..0.997 rows=445 loops=1)
                -> Filter: (o.topLayer1 is not null)  (cost=45.2 rows=445) (actual time=0.0919..0.302 rows=445 loops=1)
                    -> Table scan on o  (cost=45.2 rows=445) (actual time=0.0909..0.263 rows=445 loops=1)
                -> Single-row index lookup on c using PRIMARY (clothingId=o.topLayer1)  (cost=0.25 rows=1) (actual time=0.00128..0.00137 rows=1 loops=445)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: max(sub.style_count)  (cost=2.5..2.5 rows=1) (actual time=0.0483..0.0483 rows=1 loops=319)
        -> Table scan on sub  (cost=2.5..2.5 rows=0) (actual time=0.0472..0.0475 rows=3.38 loops=319)
            -> Materialize  (cost=0..0 rows=0) (actual time=0.0469..0.0469 rows=3.38 loops=319)
                -> Table scan on <temporary>  (actual time=0.0457..0.0462 rows=3.38 loops=319)
                    -> Aggregate using temporary table  (actual time=0.0454..0.0454 rows=3.38 loops=319)
                        -> Nested loop inner join  (cost=11.5 rows=25.3) (actual time=0.014..0.041 rows=4.57 loops=319)
                            -> Covering index lookup on c2 using idx_userid (userId=c.userId)  (cost=2.64 rows=20) (actual time=0.00505..0.00764 rows=20 loops=319)
                            -> Index lookup on o2 using idx_toplayer1 (topLayer1=c2.clothingId)  (cost=0.322 rows=1.26) (actual time=0.00143..0.00153 rows=0.229 loops=6380)
```

Index #2:
CREATE INDEX idx_id ON clothingItems(clothingId);
DROP INDEX idx_id ON clothingItems;



```
| -> Filter: (`count(0)` = `(select #2)`)  (actual time=277..277 rows=159 loops=1)
    -> Table scan on <temporary>  (actual time=277..277 rows=319 loops=1)
        -> Aggregate using temporary table  (actual time=277..277 rows=319 loops=1)
            -> Nested loop inner join  (cost=462 rows=445) (actual time=152..183 rows=445 loops=1)
                -> Filter: (o.topLayer1 is not null)  (cost=46.4 rows=445) (actual time=81..82.8 rows=445 loops=1)
                    -> Table scan on o  (cost=46.4 rows=445) (actual time=81..82.8 rows=445 loops=1)
                -> Single-row index lookup on c using PRIMARY (clothingId=o.topLayer1)  (cost=0.834 rows=1) (actual time=0.224..0.224 rows=1 loops=445)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: max(sub.style_count)  (cost=2.5..2.5 rows=1) (actual time=0.283..0.283 rows=1 loops=319)
        -> Table scan on sub  (cost=2.5..2.5 rows=0) (actual time=0.282..0.283 rows=3.38 loops=319)
            -> Materialize  (cost=0..0 rows=0) (actual time=0.282..0.282 rows=3.38 loops=319)
                -> Table scan on <temporary>  (actual time=0.28..0.28 rows=3.38 loops=319)
                    -> Aggregate using temporary table  (actual time=0.28..0.28 rows=3.38 loops=319)
                        -> Nested loop inner join  (cost=21.4 rows=25.3) (actual time=0.245..0.274 rows=4.57 loops=319)
                            -> Covering index lookup on c2 using idx_userid (userId=c.userId)  (cost=3.02 rows=20) (actual time=0.201..0.204 rows=20 loops=319)
                            -> Index lookup on o2 using idx_toplayer1 (topLayer1=c2.clothingId)  (cost=0.796 rows=1.26) (actual time=0.00325..0.00335 rows=0.229 loops=6380)
```

Index #3:
CREATE INDEX idx_userid_clothingid ON clothingItems(userId, clothingId);
DROP INDEX idx_userid_clothingid ON clothingItems;



```
| -> Filter: (`count(0)` = `(select #2)`)  (actual time=16.9..17 rows=159 loops=1)
    -> Table scan on <temporary>  (actual time=16.9..17 rows=319 loops=1)
        -> Aggregate using temporary table  (actual time=16.9..16.9 rows=319 loops=1)
            -> Nested loop inner join  (cost=201 rows=445) (actual time=0.188..1.26 rows=445 loops=1)
                -> Filter: (o.topLayer1 is not null)  (cost=45.2 rows=445) (actual time=0.146..0.423 rows=445 loops=1)
                    -> Table scan on o  (cost=45.2 rows=445) (actual time=0.143..0.382 rows=445 loops=1)
                -> Single-row index lookup on c using PRIMARY (clothingId=o.topLayer1)  (cost=0.25 rows=1) (actual time=0.00166..0.00169 rows=1 loops=445)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: max(sub.style_count)  (cost=2.5..2.5 rows=1) (actual time=0.0464..0.0465 rows=1 loops=319)
        -> Table scan on sub  (cost=2.5..2.5 rows=0) (actual time=0.0454..0.0457 rows=3.38 loops=319)
            -> Materialize  (cost=0..0 rows=0) (actual time=0.0452..0.0452 rows=3.38 loops=319)
                -> Table scan on <temporary>  (actual time=0.0441..0.0443 rows=3.38 loops=319)
                    -> Aggregate using temporary table  (actual time=0.0439..0.0439 rows=3.38 loops=319)
                        -> Nested loop inner join  (cost=11.5 rows=25.3) (actual time=0.014..0.0396 rows=4.57 loops=319)
                            -> Covering index lookup on c2 using idx_userid (userId=c.userId)  (cost=2.64 rows=20) (actual time=0.00562..0.0081 rows=20 loops=319)
                            -> Index lookup on o2 using idx_toplayer1 (topLayer1=c2.clothingId)  (cost=0.322 rows=1.26) (actual time=0.00136..0.00145 rows=0.229 loops=6380)
1 row in set, 2 warnings (0.04 sec)
```

	Cost
Default	201

idx_toplayer1	201
idx_id	462
idx_userid_clothingid	201

We tried indexes on the clothingItems and outfits tables, as well as both of them together(on their join and group by clauses). This did not benefit the cost at all and actually was harmful in some cases. We ultimately decided that the default index worked the best.

QUERY #3

```
SELECT u.userId, o.outfitId, t1.name AS topLayer1_name, t2.name AS topLayer2_name, b.name AS bottom_name, w.dailyAvg, w.location
FROM users u
JOIN weather w ON u.location = w.location AND w.month = 1 AND w.day = 1
JOIN outfits o ON o.outfitId = (
   SELECT o2.outfitId
   FROM outfits o2
   JOIN clothingItems t1 ON o2.topLayer1 = t1.clothingId AND t1.userId = u.userId
   LEFT JOIN clothingItems t2 ON o2.topLayer2 = t2.clothingId AND t2.userId = u.userId
   JOIN clothingItems b ON o2.bottom = b.clothingId AND b.userId = u.userId
   WHERE (
      (w.dailyAvg < 35 AND (t1.type = 'jacket' OR IFNULL(t2.type, '') = 'jacket')) OR
            (w.dailyAvg >= 35 AND w.dailyAvg < 45 AND (t1.type = 'long sleeve' OR IFNULL(t2.type, '') = 'long sleeve')) OR
      (w.dailyAvg >= 45 AND (
         t1.type IN ('jacket', 'long sleeve') OR
         IFNULL(t2.type, '') IN ('jacket', 'long sleeve')
      ))
   )
   LIMIT 1
)
JOIN clothingItems t1 ON o.topLayer1 = t1.clothingId AND t1.userId = u.userId
LEFT JOIN clothingItems t2 ON o.topLayer2 = t2.clothingId AND t2.userId = u.userId
JOIN clothingItems b ON o.bottom = b.clothingId AND b.userId = u.userId
ORDER BY userId ASC;
```

This is a more advanced query that selects a user's outfit based on the weather, at their location, at that specific date. If the weather is less than 35 degrees, it picks an outfit with a jacket. If the weather is within the range of 35 to 45 degrees, it picks an outfit with a jacket or long sleeve. If it

is greater than 45, it will select any outfit. This will be a very helpful query to actually determine what outfit to recommend to the user based on the weather.

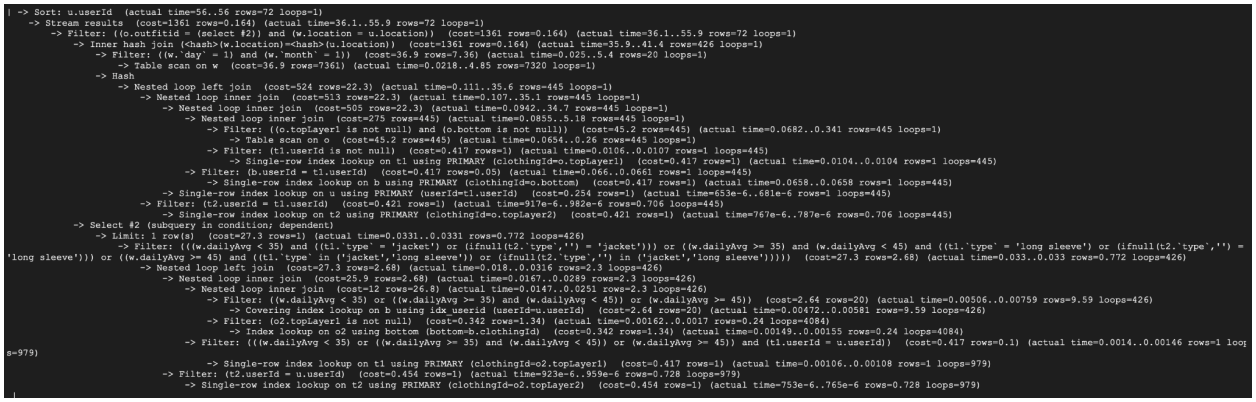
```
mysql> SELECT      u.userId,     o.outfitId,     t1.name AS topLayer1_name,     t2.name AS topLayer2_name,     b.name AS bottom_name,     w.dailyAvg,     w.location FROM users u JOIN weather w ON u.location = w.location ANI
 w.month = 1 AND w.day = 1 JOIN outfits o ON o.outfitId = (     SELECT o2.outfitId     FROM outfits o2     JOIN clothingItems t1 ON o2.topLayer1 = t1.clothingId AND t1.userId = u.userId     LEFT JOIN clothingItems t2 ON o2.
topLayer2 = t2.clothingId AND t2.userId = u.userId     JOIN clothingItems b ON o2.bottom = b.clothingId AND b.userId = u.userId     WHERE (         (w.dailyAvg < 35 AND (t1.type = 'jacket' OR IFNULL(t2.type, '') = 'jacket')
) OR         (w.dailyAvg >= 35 AND w.dailyAvg < 45 AND (t1.type = 'long sleeve' OR IFNULL(t2.type, '') = 'long sleeve')) OR         (w.dailyAvg >= 45 AND (                 t1.type IN ('jacket', 'long sleeve') OR IFN
ULL(t2.type, '') IN ('jacket', 'long sleeve')         ))     )     LIMIT 1 ) JOIN clothingItems t1 ON o.topLayer1 = t1.clothingId AND t1.userId = u.userId LEFT JOIN clothingItems t2 ON o.topLayer2 = t2.clothingId AND t2.use
rId = u.userId JOIN clothingItems b ON o.bottom = b.clothingId AND b.userId = u.userId ORDER BY userId ASC LIMIT 15;
+--------+---------+--------------------+--------------------+--------------+---------+------------------+
| userId | outfitId | topLayer1_name     | topLayer2_name     | bottom_name  | dailyAvg | location         |
+--------+---------+--------------------+--------------------+--------------+---------+------------------+
|      1 |     7001 | green dress shirt  | green jacket       | yellow jeans |     34.5 | New York, NY     |
|      2 |     7006 | red sweater        | gray jacket        | beige jeans  |     46.2 | Dallas, TX       |
|      3 |     7010 | green long sleeve  | beige rain coat    | white pants  |     50.6 | San Francisco, CA |
|      4 |     7015 | gray long sleeve   | NULL               | red jeans    |     50.6 | San Francisco, CA |
|      5 |     7019 | blue long sleeve   | brown jacket       | navy jeans   |     49.5 | Austin, TX       |
|      6 |     7024 | blue long sleeve   | beige rain coat    | blue jeans   |     46.5 | Las Vegas, NV    |
|      7 |     7029 | brown long sleeve  | red jacket         | red jeans    |     41.9 | Seattle, WA      |
|      9 |     7038 | blue long sleeve   | NULL               | blue pants   |     60.2 | Los Angeles, CA  |
|     10 |     7041 | white long sleeve  | red rain coat      | yellow shorts |     53.3 | Phoenix, AZ      |
|     11 |     7044 | brown long sleeve  | red rain coat      | white pants  |     60.2 | Los Angeles, CA  |
|     13 |     7053 | brown long sleeve  | green jacket       | blue pants   |     46.2 | Dallas, TX       |
|     15 |     7064 | green dress shirt  | gray jacket        | brown pants  |     30.7 | Detroit, MI      |
|     17 |     7073 | green dress shirt  | brown jacket       | yellow jeans |     71.3 | Miami, Fl        |
|     19 |     7084 | green t-shirt      | beige jacket       | green jeans  |     46.2 | Dallas, TX       |
|     20 |     7086 | yellow dress shirt | white jacket       | blue pants   |     17.9 | Minneapolis, MN  |
+--------+---------+--------------------+--------------------+--------------+---------+------------------+
15 rows in set (0.02 sec)
```

```
| -> Sort: u.userId  (actual time=333..333 rows=72 loops=1)
    -> Stream results  (cost=1226 rows=0.164) (actual time=268..332 rows=72 loops=1)
        -> Filter: ((o.outfitId = (select #2)) and (w.location = u.location))  (cost=1226 rows=0.164) (actual time=268..332 rows=72 loops=1)
            -> Inner hash join (<hash>(w.location)=<hash>(u.location))  (cost=1226 rows=0.164) (actual time=246..294 rows=426 loops=1)
                -> Filter: ((w.`day` = 1) and (w.`month` = 1))  (cost=36.9 rows=7.36) (actual time=127..175 rows=20 loops=1)
                    -> Table scan on w  (cost=36.9 rows=7361) (actual time=127..175 rows=7320 loops=1)
                -> Hash
                    -> Nested loop left join  (cost=389 rows=22.3) (actual time=114..118 rows=445 loops=1)
                        -> Nested loop inner join  (cost=381 rows=22.3) (actual time=114..118 rows=445 loops=1)
                            -> Nested loop inner join  (cost=357 rows=22.3) (actual time=0.856..2.63 rows=445 loops=1)
                                -> Nested loop inner join  (cost=201 rows=445) (actual time=0.844..1.99 rows=445 loops=1)
                                    -> Filter: ((o.topLayer1 is not null) and (o.bottom is not null))  (cost=45.2 rows=445) (actual time=0.816..1.02 rows=445 loops=1)
                                        -> Table scan on o  (cost=45.2 rows=445) (actual time=0.111..0.256 rows=445 loops=1)
                                    -> Filter: (t1.userId is not null)  (cost=0.25 rows=1) (actual time=0.00191..0.002 rows=1 loops=445)
                                        -> Single-row index lookup on t1 using PRIMARY (clothingId=o.topLayer1)  (cost=0.25 rows=1) (actual time=0.00176..0.00179 rows=1 loops=445)
                                -> Filter: (b.userId = t1.userId)  (cost=0.25 rows=0.05) (actual time=0.00119..0.00128 rows=1 loops=445)
                                    -> Single-row index lookup on b using PRIMARY (clothingId=o.bottom)  (cost=0.25 rows=1) (actual time=0.00102..0.00105 rows=1 loops=445)
                            -> Single-row index lookup on u using PRIMARY (userId=t1.userId)  (cost=1 rows=1) (actual time=0.258..0.258 rows=1 loops=445)
                        -> Filter: (t2.userId = t1.userId)  (cost=0.254 rows=1) (actual time=0.00104..0.0011 rows=0.706 loops=445)
                            -> Single-row index lookup on t2 using PRIMARY (clothingId=o.topLayer2)  (cost=0.254 rows=1) (actual time=849e-6..869e-6 rows=0.706 loops=445)
                -> Select #2 (subquery in condition; dependent)
                    -> Limit: 1 row(s)  (cost=22.3 rows=1) (actual time=0.0872..0.0872 rows=0.772 loops=426)
                        -> Filter: (((w.dailyAvg < 35) and ((t1.`type` = 'jacket') or (ifnull(t2.`type`,'') = 'jacket'))) or ((w.dailyAvg >= 35) and (w.dailyAvg < 45) and ((t1.`type` = 'long sleeve') or (ifnull(t2.`type`,'') =
'long sleeve'))) or ((w.dailyAvg >= 45) and ((t1.`type` in ('jacket','long sleeve')) or (ifnull(t2.`type`,'') in ('jacket','long sleeve')))))  (cost=22.3 rows=2.68) (actual time=0.087..0.087 rows=0.772 loops=426)
                            -> Nested loop left join  (cost=22.3 rows=2.68) (actual time=0.0718..0.0854 rows=2.3 loops=426)
                                -> Nested loop inner join  (cost=21.4 rows=2.68) (actual time=0.0703..0.0825 rows=2.3 loops=426)
                                    -> Nested loop inner join  (cost=12 rows=26.8) (actual time=0.0679..0.0782 rows=2.3 loops=426)
                                        -> Filter: ((w.dailyAvg < 35) or ((w.dailyAvg >= 35) and (w.dailyAvg < 45)) or (w.dailyAvg >= 45))  (cost=2.64 rows=20) (actual time=0.0069..0.00952 rows=9.59 loops=426)
                                            -> Covering index lookup on b using idx_userid (userId=u.userId)  (cost=2.64 rows=20) (actual time=0.00625..0.00738 rows=9.59 loops=426)
                                        -> Filter: (o2.topLayer1 is not null)  (cost=0.342 rows=1.34) (actual time=0.00695..0.00702 rows=0.24 loops=4084)
                                            -> Index lookup on o2 using bottom (bottom=b.clothingId)  (cost=0.342 rows=1.34) (actual time=0.00681..0.00687 rows=0.24 loops=4084)
                                    -> Filter: (((w.dailyAvg < 35) or ((w.dailyAvg >= 35) and (w.dailyAvg < 45)) or (w.dailyAvg >= 45)) and (t1.userId = u.userId))  (cost=0.25 rows=0.1) (actual time=0.00161..0.00168 rows=1 loop
s=979)
                                        -> Single-row index lookup on t1 using PRIMARY (clothingId=o2.topLayer1)  (cost=0.25 rows=1) (actual time=0.00125..0.00127 rows=1 loops=979)
                                -> Filter: (t2.userId = u.userId)  (cost=0.287 rows=1) (actual time=974e-6..0.00103 rows=0.728 loops=979)
                                    -> Single-row index lookup on t2 using PRIMARY (clothingId=o2.topLayer2)  (cost=0.287 rows=1) (actual time=805e-6..817e-6 rows=0.728 loops=979)
|
```

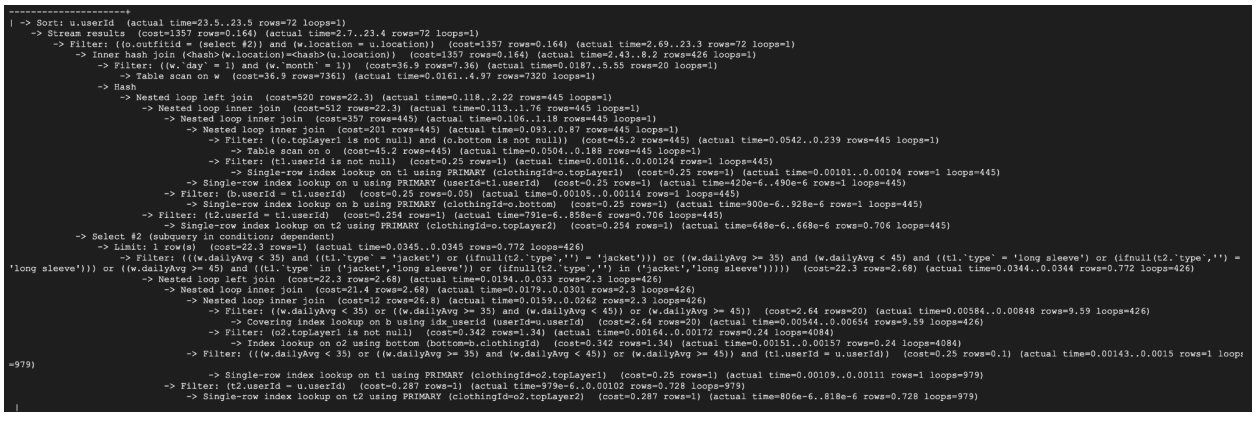
Index #1:
CREATE INDEX idx_loc_date ON weather(location, month, day);
DROP INDEX idx_date ON weather;

```
| -> Nested loop left join  (cost=162 rows=0.251) (actual time=32.3..53.8 rows=72 loops=1)
    -> Nested loop inner join  (cost=162 rows=0.251) (actual time=32.3..53.6 rows=72 loops=1)
        -> Nested loop inner join  (cost=158 rows=5.03) (actual time=32.1..53.3 rows=72 loops=1)
            -> Nested loop inner join  (cost=80.6 rows=101) (actual time=32.1..53.1 rows=72 loops=1)
                -> Nested loop inner join  (cost=45.4 rows=101) (actual time=0.312..1.41 rows=96 loops=1)
                    -> Index scan on u using PRIMARY  (cost=10.2 rows=100) (actual time=0.26..0.333 rows=100 loops=1)
                    -> Index lookup on w using idx_weather_loc_date (location=u.location, month=1, day=1)  (cost=0.252 rows=1.01) (actual time=0.00944..0.0105 rows=0.96 loops=100)
                -> Filter: ((o.outfitId = (select #2)) and (o.topLayer1 is not null) and (o.bottom is not null))  (cost=0.251 rows=1) (actual time=0.538..0.539 rows=0.75 loops=96)
                    -> Single-row index lookup on o using PRIMARY (outfitId=(select #2))  (cost=0.251 rows=1) (actual time=0.517..0.517 rows=0.75 loops=96)
                    -> Select #2 (subquery in condition; dependent)
                        -> Limit: 1 row(s)  (cost=34.6 rows=1) (actual time=0.214..0.214 rows=0.9 loops=240)
                            -> Filter: (((w.dailyAvg < 35) and ((t1.`type` = 'jacket') or (ifnull(t2.`type`,'') = 'jacket'))) or ((w.dailyAvg >= 35) and (w.dailyAvg < 45) and ((t1.`type` = 'long sleeve') or (ifnull(t2.`type
`,'') = 'long sleeve'))) or ((w.dailyAvg >= 45) and ((t1.`type` in ('jacket','long sleeve')) or (ifnull(t2.`type`,'') in ('jacket','long sleeve')))))  (cost=34.6 rows=2.68) (actual time=0.214..0.214 rows=0.9 loops=240)
                                -> Nested loop left join  (cost=34.6 rows=2.68) (actual time=0.202..0.212 rows=1.9 loops=240)
                                    -> Nested loop inner join  (cost=32.6 rows=2.68) (actual time=0.2..0.209 rows=1.9 loops=240)
                                        -> Nested loop inner join  (cost=12 rows=26.8) (actual time=0.0155..0.0225 rows=1.9 loops=240)
                                            -> Filter: ((w.dailyAvg < 35) or ((w.dailyAvg >= 35) and (w.dailyAvg < 45)) or (w.dailyAvg >= 45))  (cost=2.64 rows=20) (actual time=0.00536..0.0074 rows=8 loops=240)
                                                -> Covering index lookup on b using idx_userid (userId=u.userId)  (cost=2.64 rows=20) (actual time=0.00502..0.00588 rows=8 loops=240)
                                            -> Filter: (o2.topLayer1 is not null)  (cost=0.342 rows=1.34) (actual time=0.0017..0.00177 rows=0.238 loops=1920)
                                                -> Index lookup on o2 using bottom (bottom=b.clothingId)  (cost=0.342 rows=1.34) (actual time=0.00157..0.00163 rows=0.238 loops=1920)
                                        -> Filter: (((w.dailyAvg < 35) or ((w.dailyAvg >= 35) and (w.dailyAvg < 45)) or (w.dailyAvg >= 45)) and (t1.userId = u.userId))  (cost=0.667 rows=0.1) (actual time=0.0975..0.0975 row
=1 loops=457)
                                            -> Single-row index lookup on t1 using PRIMARY (clothingId=o2.topLayer1)  (cost=0.667 rows=1) (actual time=0.097..0.0971 rows=1 loops=457)
                                    -> Filter: (t2.userId = u.userId)  (cost=0.704 rows=1) (actual time=0.00172..0.00175 rows=0.737 loops=457)
                                        -> Single-row index lookup on t2 using PRIMARY (clothingId=o2.topLayer2)  (cost=0.704 rows=1) (actual time=0.00151..0.00152 rows=0.737 loops=457)
            -> Filter: (t1.userId = u.userId)  (cost=0.667 rows=0.05) (actual time=0.00246..0.00255 rows=1 loops=72)
                -> Single-row index lookup on t1 using PRIMARY (clothingId=o.topLayer1)  (cost=0.667 rows=1) (actual time=0.00223..0.00226 rows=1 loops=72)
        -> Filter: (b.userId = u.userId)  (cost=0.668 rows=0.05) (actual time=0.00409..0.00419 rows=1 loops=72)
            -> Single-row index lookup on b using PRIMARY (clothingId=o.bottom)  (cost=0.668 rows=1) (actual time=0.00392..0.00394 rows=1 loops=72)
    -> Filter: (t2.userId = u.userId)  (cost=1.06 rows=1) (actual time=0.00122..0.00131 rows=0.917 loops=72)
        -> Single-row index lookup on t2 using PRIMARY (clothingId=o.topLayer2)  (cost=1.06 rows=1) (actual time=0.00104..0.00107 rows=0.917 loops=72)
|
```

Index #2:
CREATE INDEX idx_outfitid ON outfits(outfitId);
DROP INDEX idx_outfitid ON outfits;



Index #3:
CREATE INDEX idx_userid_type ON clothingItems(userId, type);
DROP INDEX idx_userid_type ON clothingItems;



	Cost
Default	1226
idx_loc_date	162
idx_outfitid	1361
idx_userid_type	1367

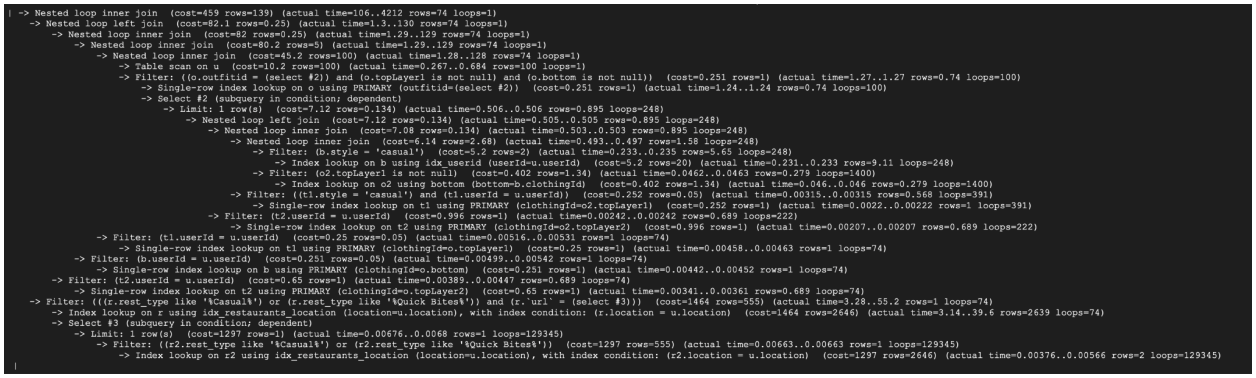
We put indexes on the weather, clothingItems, and outfits table(join clauses). The only beneficial one was the index on the weather join. This significantly decreased the cost, so this was out best outcome.

QUERY #4

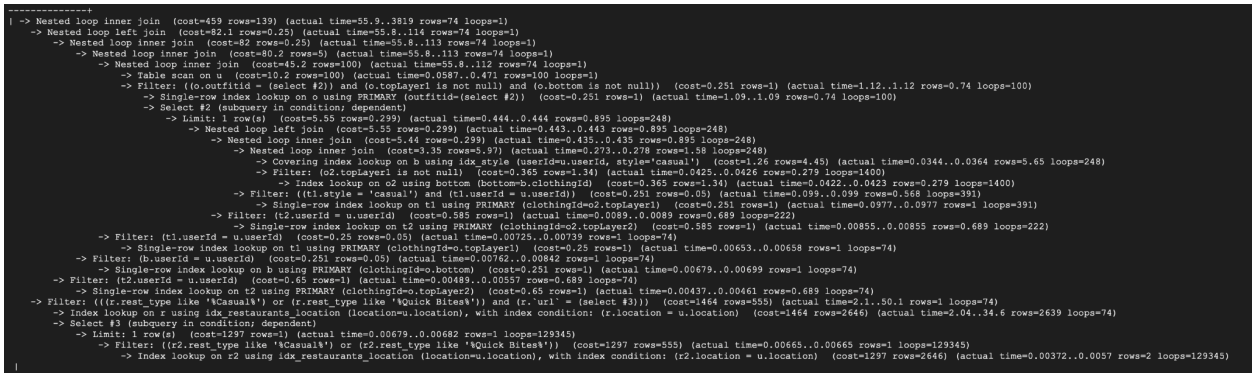
```
SELECT u.userId, u.location, o.outfitId, t1.name AS topLayer1_name, t2.name AS topLayer2_name, b.name AS bottom_name, r.name AS restaurant_name, r.rest_type
FROM users u
JOIN outfits o ON o.outfitId = (
    SELECT o2.outfitId
    FROM outfits o2
    JOIN clothingItems t1 ON o2.topLayer1 = t1.clothingId AND t1.userId = u.userId
    LEFT JOIN clothingItems t2 ON o2.topLayer2 = t2.clothingId AND t2.userId = u.userId
    JOIN clothingItems b ON o2.bottom = b.clothingId AND b.userId = u.userId
    WHERE t1.style = 'casual' AND b.style = 'casual'
    LIMIT 1
)
JOIN clothingItems t1 ON o.topLayer1 = t1.clothingId AND t1.userId = u.userId
LEFT JOIN clothingItems t2 ON o.topLayer2 = t2.clothingId AND t2.userId = u.userId
JOIN clothingItems b ON o.bottom = b.clothingId AND b.userId = u.userId
JOIN restaurants r ON r.location = u.location
WHERE (r.rest_type LIKE '%Casual%' OR r.rest_type LIKE '%Quick Bites%')
  AND r.url = (
    SELECT r2.url
    FROM restaurants r2
    WHERE r2.location = u.location
      AND (r2.rest_type LIKE '%Casual%' OR r2.rest_type LIKE '%Quick Bites%')
    LIMIT 1
);
```

This is another advanced query that suggests an outfit for a user, at their location, which matches the style to the restaurant. This query will also be a key component to recommending restaurants. It can be further changed to select all sorts of styles and types of restaurants but this is just an example with casual mapping to quick bites or casual dining restaurants.

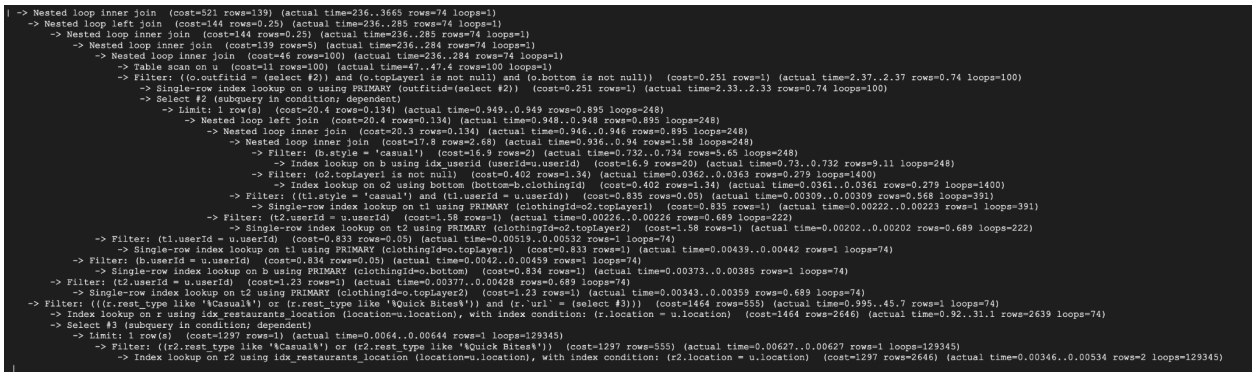
```
mysql> SELECT     u.userId,     u.location,     o.outfitId,     t1.name AS topLayer1_name,     t2.name AS topLayer2_name,     b.name AS bottom_name,     r.name AS restaurant_name,     r.rest_type FROM users u JOIN outfits
o ON o.outfitId = (     SELECT o2.outfitId     FROM outfits o2     JOIN clothingItems t1 ON o2.topLayer1 = t1.clothingId AND t1.userId = u.userId     LEFT JOIN clothingItems t2 ON o2.topLayer2 = t2.clothingId AND t2.userId
= u.userId     JOIN clothingItems b ON o2.bottom = b.clothingId AND b.userId = u.userId     WHERE t1.style = 'casual' AND b.style = 'casual'     LIMIT 1 ) JOIN clothingItems t1 ON o.topLayer1 = t1.clothingId AND t1.userId =
 u.userId LEFT JOIN clothingItems t2 ON o.topLayer2 = t2.clothingId AND t2.userId = u.userId JOIN clothingItems b ON o.bottom = b.clothingId AND b.userId = u.userId JOIN restaurants r ON r.location = u.location WHERE (r.res
t_type LIKE '%Casual%' OR r.rest_type LIKE '%Quick Bites%')   AND r.url = (     SELECT r2.url     FROM restaurants r2     WHERE r2.location = u.location       AND (r2.rest_type LIKE '%Casual%' OR r2.rest_type LIKE '%Quick B
ites%')     LIMIT 1 ) LIMIT 15;
+--------+-----------------+---------+-----------------+-----------------+-------------+-----------------------------------+-------------+
| userId | location        | outfitId | topLayer1_name  | topLayer2_name  | bottom_name | restaurant_name                   | rest_type   |
+--------+-----------------+---------+-----------------+-----------------+-------------+-----------------------------------+-------------+
|      1 | New York, NY    |    7002 | red t-shirt     | green jacket    | yellow jeans | Jalsa                            | Casual Dining |
|      2 | Dallas, TX      |    7007 | brown t-shirt   | NULL            | green pants | Beijing Bites                     | Casual Dining |
|      3 | San Francisco, CA |    7010 | green long sleeve | beige rain coat | white pants | Tejas Bar and Restaurant          | Casual Dining |
|      4 | San Francisco, CA |    7013 | navy t-shirt    | NULL            | yellow jeans | Tejas Bar and Restaurant          | Casual Dining |
|      5 | Austin, TX      |    7019 | blue long sleeve | brown jacket    | navy jeans  | Pizza Palace                      | Quick Bites   |
|      6 | Las Vegas, NV   |    7024 | blue long sleeve | beige rain coat | blue jeans  | Tree Top                          | Casual Dining |
|      7 | Seattle, WA     |    7030 | green t-shirt   | yellow rain coat | blue pants  | Donne Biriyani House              | Quick Bites   |
|      9 | Los Angeles, CA |    7038 | blue long sleeve | NULL            | blue pants  | Barbeque Nation                   | Casual Dining |
|     10 | Phoenix, AZ     |    7040 | brown t-shirt   | red rain coat   | beige pants | Curry with a 'K' - St. Mark's Hotel | Casual Dining |
|     11 | Los Angeles, CA |    7044 | brown long sleeve | red rain coat   | white pants | Barbeque Nation                   | Casual Dining |
|     12 | Detroit, MI     |    7049 | gray t-shirt    | white rain coat | white jeans | Kerala Pavilion                   | Quick Bites   |
|     13 | Dallas, TX      |    7053 | brown long sleeve | green jacket    | blue pants  | Beijing Bites                     | Casual Dining |
|     14 | Minneapolis, MN |    7057 | navy t-shirt    | NULL            | gray jeans  | Roti Shotti                       | Quick Bites   |
|     15 | Detroit, MI     |    7066 | green long sleeve | white rain coat | navy jeans  | Kerala Pavilion                   | Quick Bites   |
|     16 | Houston, TX     |    7068 | beige t-shirt   | blue jacket     | red jeans   | Ebony                             | Casual Dining |
+--------+-----------------+---------+-----------------+-----------------+-------------+-----------------------------------+-------------+
15 rows in set (1.89 sec)
```



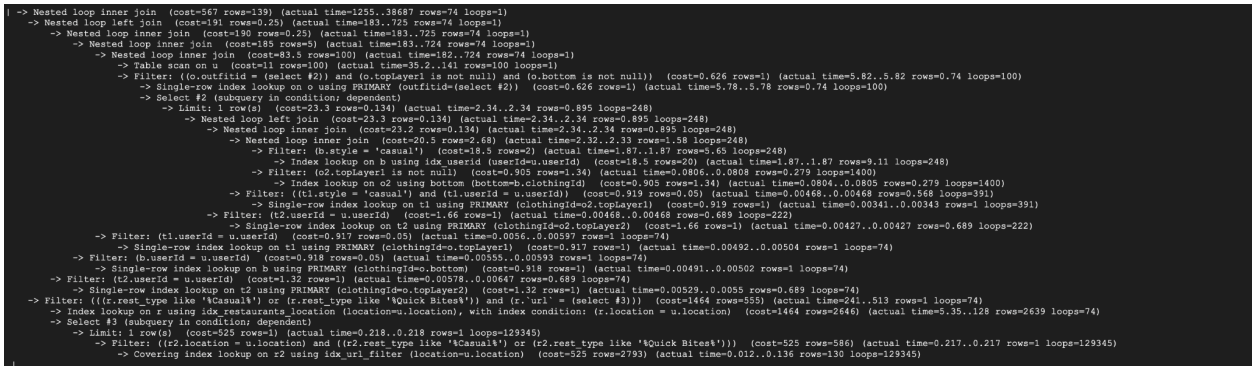
Index #1:
CREATE INDEX idx_style ON clothingItems(userId, style);
DROP INDEX idx_style ON clothingItems;



Index #2:
CREATE INDEX idx_restaurants_type ON restaurants(location, rest_type);
DROP INDEX idx_restaurants_type ON restaurants;



Index #3:
CREATE INDEX idx_url_filter ON restaurants(location, rest_type, url);
DROP INDEX idx_url_filter ON restaurants;



	Cost
Default	459
idx_style	459
idx_restaurants_type	521
idx_url_filter	567

We put indexes on the clothingItems and restaurants tables(join clauses and group by). We saw an increase in cost for most and no change for the clothingItems index. Wwe ultimately decided that the default index was our best choice.