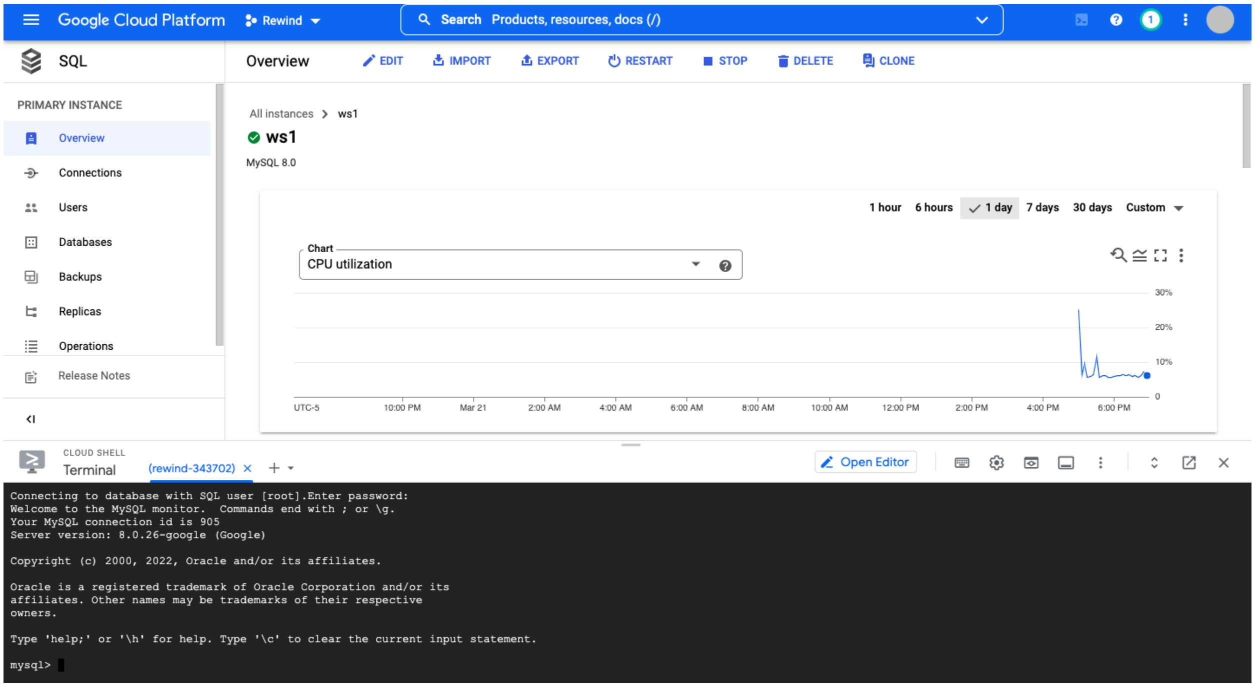
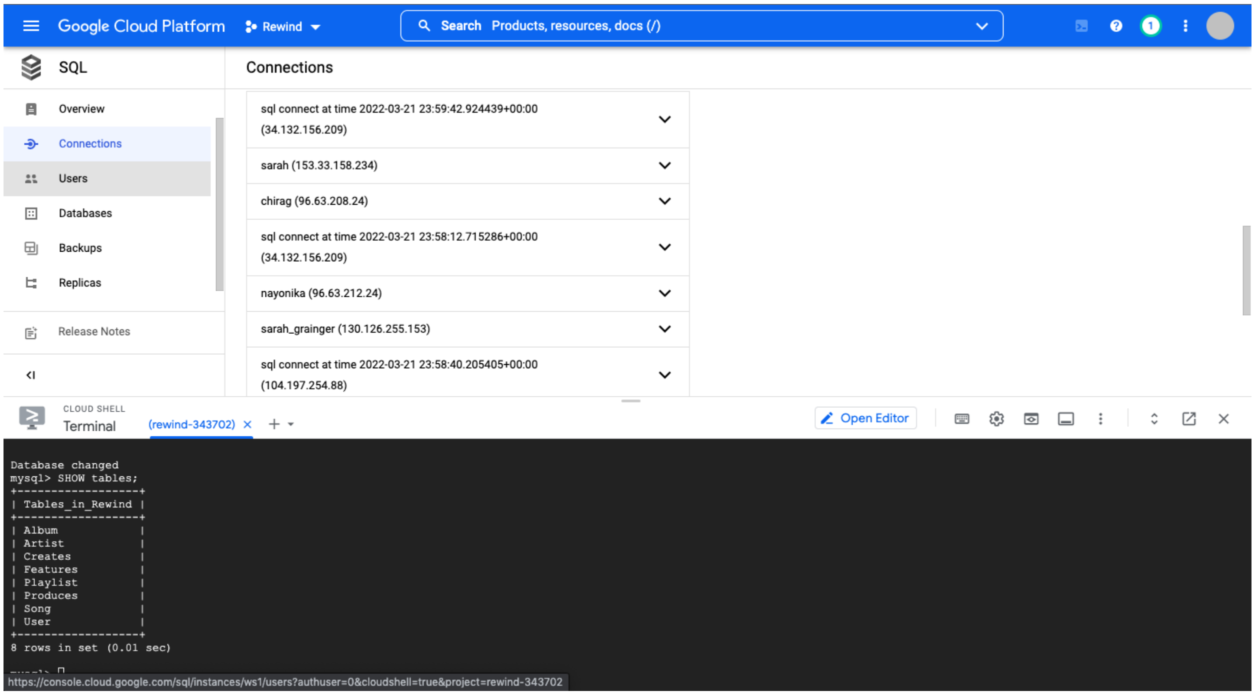
STAGE 3

1. Database Implementation

a. Connections and importing of tables



```
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 905
Server version: 8.0.26-google (Google)

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

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

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

mysql> 
```



sql connect at time 2022-03-21 23:59:42.924439+00:00 (34.132.156.209)

sarah (153.33.158.234)

chirag (96.63.208.24)

sql connect at time 2022-03-21 23:58:12.715286+00:00 (34.132.156.209)

nayonika (96.63.212.24)

sarah_grainger (130.126.255.153)

sql connect at time 2022-03-21 23:58:40.205405+00:00 (104.197.254.88)

```
Database changed
mysql> SHOW tables;
+------------------+
| Tables_in_Rewind |
+------------------+
| Album            |
| Artist           |
| Creates          |
| Features         |
| Playlist         |
| Produces         |
| Song             |
| User             |
+------------------+
8 rows in set (0.01 sec)

mysql> 
```

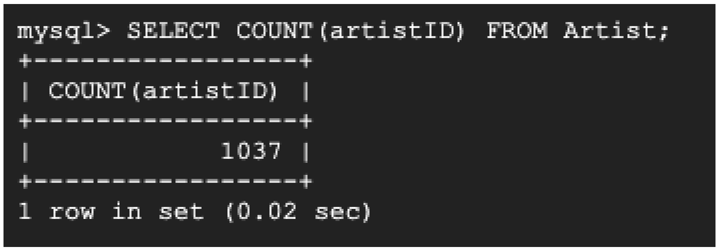
https://console.cloud.google.com/sql/instances/ws1/users?authuser=0&cloudshell=true&project=rewind-343702

b. DDL commands

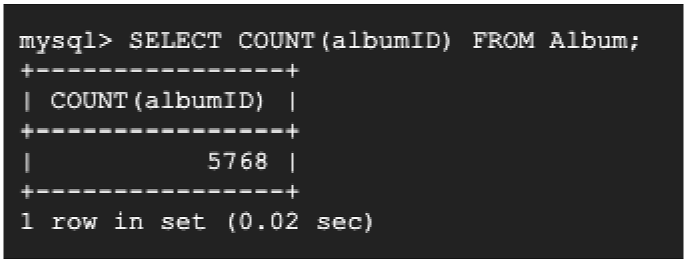
```
CREATE TABLE Artist (
        artistID VARCHAR(50) PRIMARY KEY,
        name VARCHAR(50) NOT NULL,
        followers INTEGER,
        image VARCHAR(150),
        popularityRating INT
 );

CREATE TABLE Song (
        songID VARCHAR(50) PRIMARY KEY,
        name VARCHAR(50) NOT NULL,
        genre VARCHAR(50),
        popularity INTEGER,
        releaseDate DATE,
        totalDuration FLOAT,
        albumID VARCHAR(50),
        FOREIGN KEY (albumID) REFERENCES Album(albumID) ON DELETE SET NULL
);

CREATE TABLE Playlist (
        playlistID VARCHAR(50) PRIMARY KEY,
        link VARCHAR(1000),
        numSongs INTEGER,
        minYear INTEGER,
        maxYear INTEGER,
        title VARCHAR(1000),
        totalDuration FLOAT,
        userID VARCHAR(50),
        FOREIGN KEY (userID) REFERENCES User(userID) ON DELETE CASCADE
);

CREATE TABLE User (
        userID VARCHAR(50) PRIMARY KEY,
        firstname VARCHAR(20),
        lastname VARCHAR(20),
        queries INTEGER,
        lastLogin DATE
);
```

```
CREATE TABLE Album (
        albumID VARCHAR(50) PRIMARY KEY,
        name VARCHAR(50) NOT NULL,
        genre VARCHAR(50),
        popularity INTEGER,
        releaseDate DATE,
        numSongs INTEGER,
        totalDuration FLOAT
);

CREATE TABLE Creates (
        artistID VARCHAR(50),
        songID VARCHAR(50),
        PRIMARY KEY (artistID,songID),
        FOREIGN KEY (artistID) REFERENCES Artist(artistID),
        FOREIGN KEY (songID) REFERENCES Song(SongID)
);

CREATE TABLE Produces (
        artistID VARCHAR(50),
        albumID VARCHAR(50),
        PRIMARY KEY (artistID,albumID),
        FOREIGN KEY (artistID) REFERENCES Artist(artistID),
        FOREIGN KEY (albumID) REFERENCES Album(albumID)
);

CREATE TABLE Features (
        playlistID VARCHAR(50),
        songID VARCHAR(50),
        PRIMARY KEY (playlistID,songID),
        FOREIGN KEY (playlistID) REFERENCES Playlist(playlistID),
        FOREIGN KEY (songID) REFERENCES Song(SongID)
);
```

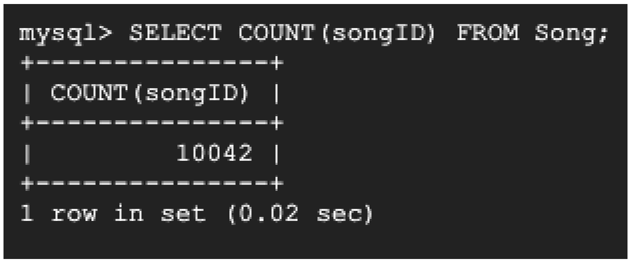
c. Count of each table:



```
mysql> SELECT COUNT(artistID) FROM Artist;
+-----------------+
| COUNT(artistID) |
+-----------------+
|            1037 |
+-----------------+
1 row in set (0.02 sec)
```



```
mysql> SELECT COUNT(albumID) FROM Album;
+----------------+
| COUNT(albumID) |
+----------------+
|           5768 |
+----------------+
1 row in set (0.02 sec)
```

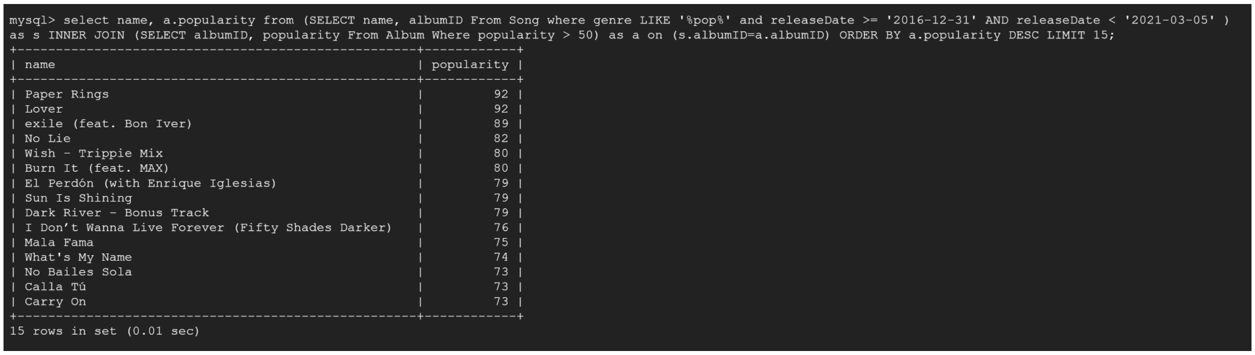


```
mysql> SELECT COUNT(songID) FROM Song;
+---------------+
| COUNT(songID) |
+---------------+
|         10042 |
+---------------+
1 row in set (0.02 sec)
```

2. Advanced Queries

SQL query to select song name and album popularity where the genre is pop, filter by release date, and choose from album with popularity rating above 50

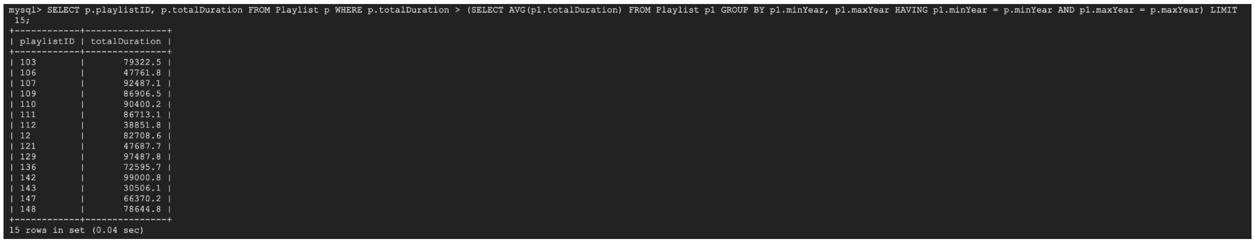
SELECT name, a.popularity
FROM (**SELECT** name, albumID
    FROM Song where genre **LIKE** '%pop%' and releaseDate >= '2016-12-31' AND releaseDate < '2021-03-05') as s
INNER JOIN (**SELECT** albumID, popularity **FROM** Album **WHERE** popularity > 50) as a
ON (s.albumID=a.albumID)
ORDER BY a.popularity DESC LIMIT 15;



```
mysql> select name, a.popularity from (SELECT name, albumID From Song where genre LIKE '%pop%' and releaseDate >= '2016-12-31' AND releaseDate < '2021-03-05' )
as s INNER JOIN (SELECT albumID, popularity From Album Where popularity > 50) as a on (s.albumID=a.albumID) ORDER BY a.popularity DESC LIMIT 15;
+----------------------------------------------------+------------+
| name                                               | popularity |
+----------------------------------------------------+------------+
| Paper Rings                                        |         92 |
| Lover                                              |         92 |
| exile (feat. Bon Iver)                             |         89 |
| No Lie                                             |         82 |
| Wish - Trippie Mix                                 |         80 |
| Burn It (feat. MAX)                                |         80 |
| El Perdón (with Enrique Iglesias)                  |         79 |
| Sun Is Shining                                     |         79 |
| Dark River - Bonus Track                           |         79 |
| I Don’t Wanna Live Forever (Fifty Shades Darker)   |         76 |
| Mala Fama                                          |         75 |
| What's My Name                                     |         74 |
| No Bailes Sola                                     |         73 |
| Calla Tú                                           |         73 |
| Carry On                                           |         73 |
+----------------------------------------------------+------------+
15 rows in set (0.01 sec)
```

SQL query to find playlists with above average playtime within time range

SELECT p.playlistID, p.totalDuration
FROM Playlist p
WHERE p.totalDuration > (**SELECT** AVG(p1.totalDuration) **FROM** Playlist p1 **GROUP BY** p1.minYear, p1.maxYear HAVING p1.minYear = p.minYear AND p1.maxYear = p.maxYear);

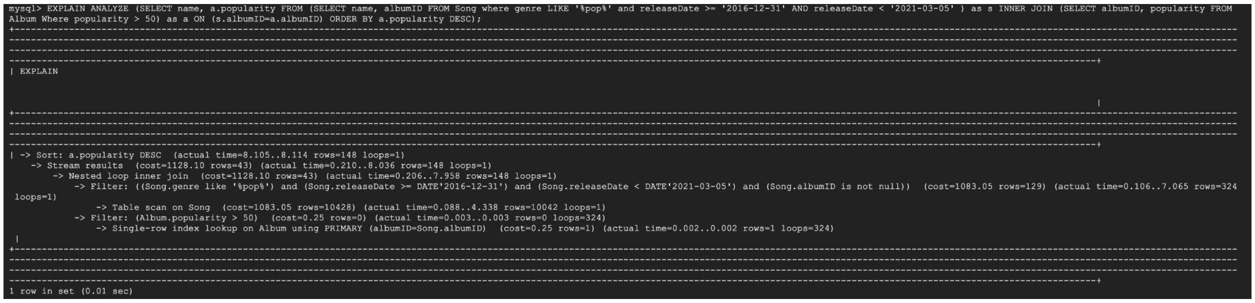


```
mysql> SELECT p.playlistID, p.totalDuration FROM Playlist p WHERE p.totalDuration > (SELECT AVG(p1.totalDuration) FROM Playlist p1 GROUP BY p1.minYear, p1.maxYear HAVING p1.minYear = p.minYear AND p1.maxYear = p.maxYear) LIMIT
 15;
+------------+---------------+
| playlistID | totalDuration |
+------------+---------------+
| 103        |       79322.5 |
| 106        |       47761.8 |
| 107        |       92487.1 |
| 109        |       86906.5 |
| 110        |       90400.2 |
| 111        |       86713.1 |
| 112        |       38851.8 |
| 12         |       82708.6 |
| 121        |       47687.7 |
| 129        |       97487.8 |
| 136        |       72595.7 |
| 142        |       99000.8 |
| 143        |       30506.1 |
| 147        |       66370.2 |
| 148        |       78644.8 |
+------------+---------------+
15 rows in set (0.04 sec)
```

3. INDEXING ANALYSIS

a) ADVANCED QUERY #1

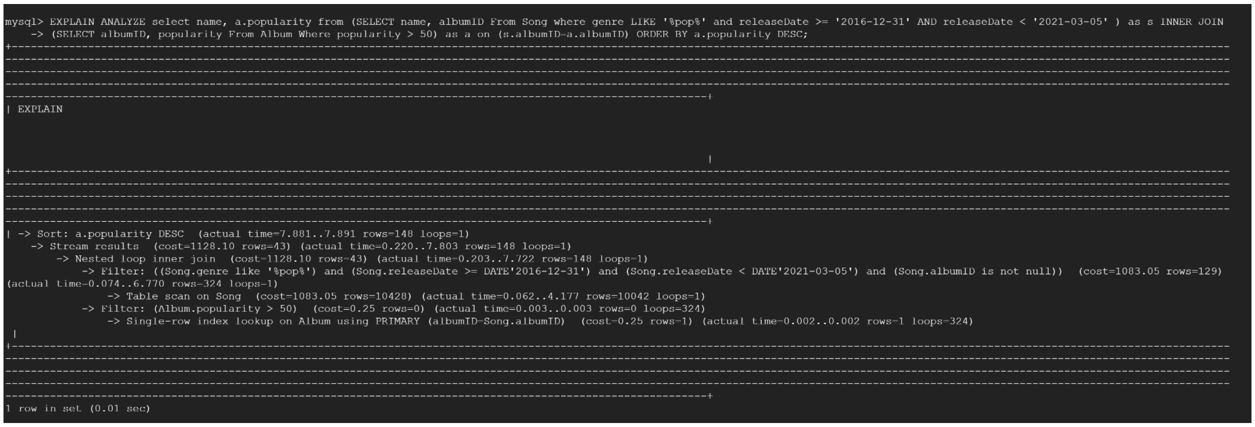
1) NO custom index:



```
mysql> EXPLAIN ANALYZE (SELECT name, a.popularity FROM (SELECT name, albumID FROM Song where genre LIKE '%pop%' and releaseDate >= '2016-12-31' AND releaseDate < '2021-03-05' ) as s INNER JOIN (SELECT albumID, popularity FROM
Album Where popularity > 50) as a ON (s.albumID=a.albumID) ORDER BY a.popularity DESC);
+------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN

                                                                                                                            |
+------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------+
| -> Sort: a.popularity DESC  (actual time=8.105..8.114 rows=148 loops=1)
    -> Stream results  (cost=1128.10 rows=43) (actual time=0.210..8.036 rows=148 loops=1)
        -> Nested loop inner join  (cost=1128.10 rows=43) (actual time=0.206..7.958 rows=148 loops=1)
            -> Filter: ((Song.genre like '%pop%') and (Song.releaseDate >= DATE'2016-12-31') and (Song.releaseDate < DATE'2021-03-05') and (Song.albumID is not null))  (cost=1083.05 rows=129) (actual time=0.106..7.065 rows=324
 loops=1)
                -> Table scan on Song  (cost=1083.05 rows=10428) (actual time=0.088..4.338 rows=10042 loops=1)
            -> Filter: (Album.popularity > 50)  (cost=0.25 rows=0) (actual time=0.003..0.003 rows=0 loops=324)
                -> Single-row index lookup on Album using PRIMARY (albumID=Song.albumID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=324)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

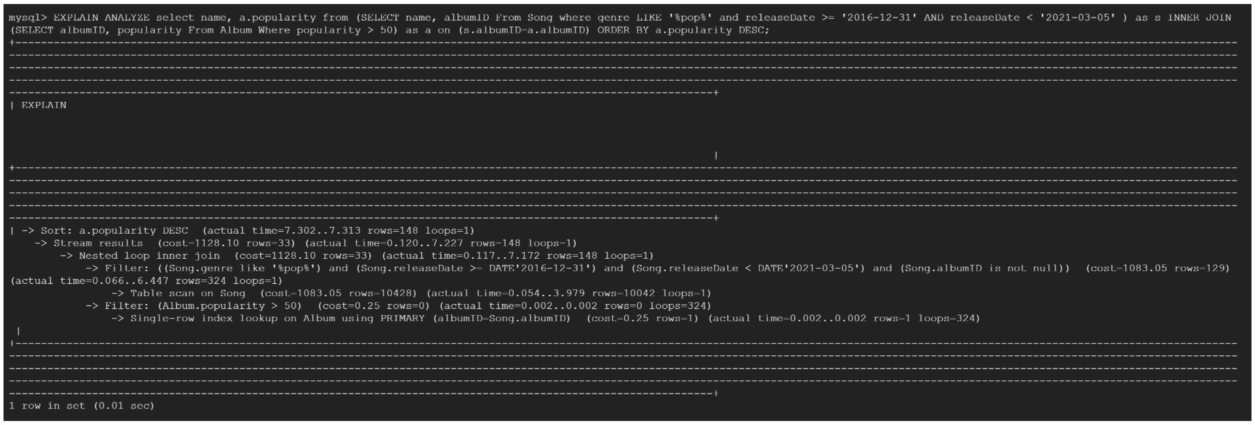
2) CREATE INDEX idx_song_genre ON Song (genre):

We see that the sort time by popularity decreases as well as the filtering time for songs by genre compared to the original_index



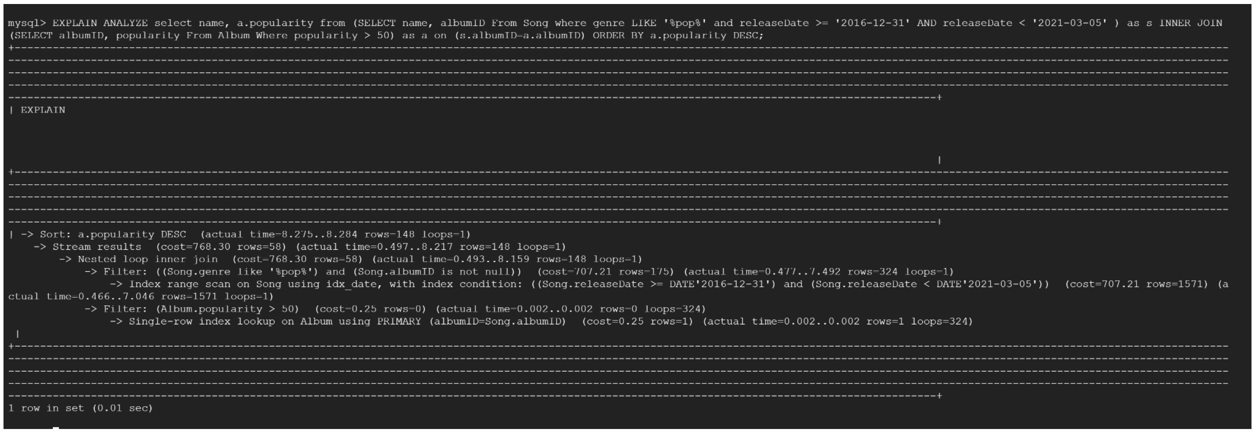
```
mysql> EXPLAIN ANALYZE select name, a.popularity from (SELECT name, albumID From Song where genre LIKE '%pop%' and releaseDate >= '2016-12-31' AND releaseDate < '2021-03-05' ) as s INNER JOIN
    -> (SELECT albumID, popularity From Album Where popularity > 50) as a on (s.albumID=a.albumID) ORDER BY a.popularity DESC;
+------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
| EXPLAIN

                                                                                    |
+------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
| -> Sort: a.popularity DESC  (actual time=7.881..7.891 rows=148 loops=1)
    -> Stream results  (cost=1128.10 rows=43) (actual time=0.220..7.803 rows=148 loops=1)
        -> Nested loop inner join  (cost=1128.10 rows=43) (actual time=0.203..7.722 rows=148 loops=1)
            -> Filter: ((Song.genre like '%pop%') and (Song.releaseDate >= DATE'2016-12-31') and (Song.releaseDate < DATE'2021-03-05') and (Song.albumID is not null))  (cost=1083.05 rows=129)
(actual time=0.074..6.770 rows=324 loops=1)
                -> Table scan on Song  (cost=1083.05 rows=10428) (actual time=0.062..4.177 rows=10042 loops=1)
            -> Filter: (Album.popularity > 50)  (cost=0.25 rows=0) (actual time=0.003..0.003 rows=0 loops=324)
                -> Single-row index lookup on Album using PRIMARY (albumID=Song.albumID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=324)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

3) CREATE INDEX idx_album_pop ON Album (popularity):
   Sorting, joining and filtering are all faster. Scanning on the song table is also faster. This is because album popularity is used in the second query. Although these processes are quicker, there is not a major impact on the runtime all together, which could possibly be because the query already runs so quickly.



```
mysql> EXPLAIN ANALYZE select name, a.popularity from (SELECT name, albumID From Song where genre LIKE '%pop%' and releaseDate >= '2016-12-31' AND releaseDate < '2021-03-05' ) as s INNER JOIN
(SELECT albumID, popularity From Album Where popularity > 50) as a on (s.albumID=a.albumID) ORDER BY a.popularity DESC;
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------+
| EXPLAIN



                                                                                                              |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------+
| -> Sort: a.popularity DESC  (actual time=7.302..7.313 rows=148 loops=1)
    -> Stream results  (cost=1128.10 rows=33) (actual time=0.120..7.227 rows=148 loops=1)
        -> Nested loop inner join  (cost=1128.10 rows=33) (actual time=0.117..7.172 rows=148 loops=1)
            -> Filter: ((Song.genre like '%pop%') and (Song.releaseDate >= DATE'2016-12-31') and (Song.releaseDate < DATE'2021-03-05') and (Song.albumID is not null))  (cost=1083.05 rows=129)
(actual time=0.066..6.447 rows=324 loops=1)
                -> Table scan on Song  (cost=1083.05 rows=10428) (actual time=0.054..3.979 rows=10042 loops=1)
            -> Filter: (Album.popularity > 50)  (cost=0.25 rows=0) (actual time=0.002..0.002 rows=0 loops=324)
                -> Single-row index lookup on Album using PRIMARY (albumID=Song.albumID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=324)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

4) CREATE INDEX idx_date ON Song (releaseDate):
   We used releaseDate as an index because we use the release date for comparison in the where clause. It was worse. We believe that unlike song_genre this comparison is an easy comparison given logical operations and that making this the index slows the query.

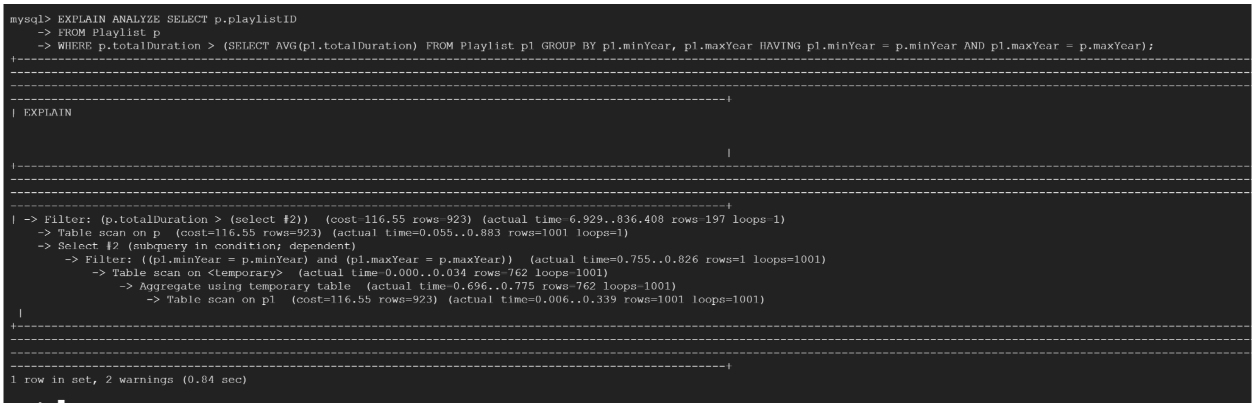


```
mysql> EXPLAIN ANALYZE select name, a.popularity from (SELECT name, albumID From Song where genre LIKE '%pop%' and releaseDate >= '2016-12-31' AND releaseDate < '2021-03-05' ) as s INNER JOIN
(SELECT albumID, popularity From Album Where popularity > 50) as a on (s.albumID=a.albumID) ORDER BY a.popularity DESC;
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN



                                                                                                                                                   |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Sort: a.popularity DESC  (actual time=8.275..8.284 rows=148 loops=1)
    -> Stream results  (cost=768.30 rows=58) (actual time=0.497..8.217 rows=148 loops=1)
        -> Nested loop inner join  (cost=768.30 rows=58) (actual time=0.493..8.159 rows=148 loops=1)
            -> Filter: ((Song.genre like '%pop%') and (Song.albumID is not null))  (cost=707.21 rows=175) (actual time=0.477..7.492 rows=324 loops=1)
                -> Index range scan on Song using idx_date, with index condition: ((Song.releaseDate >= DATE'2016-12-31') and (Song.releaseDate < DATE'2021-03-05'))  (cost=707.21 rows=1571) (a
ctual time=0.466..7.046 rows=1571 loops=1)
            -> Filter: (Album.popularity > 50)  (cost=0.25 rows=0) (actual time=0.002..0.002 rows=0 loops=324)
                -> Single-row index lookup on Album using PRIMARY (albumID=Song.albumID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=324)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Overall: indexing based on album popularity was the fastest and we chose that as our index.

b) ADVANCED QUERY #2

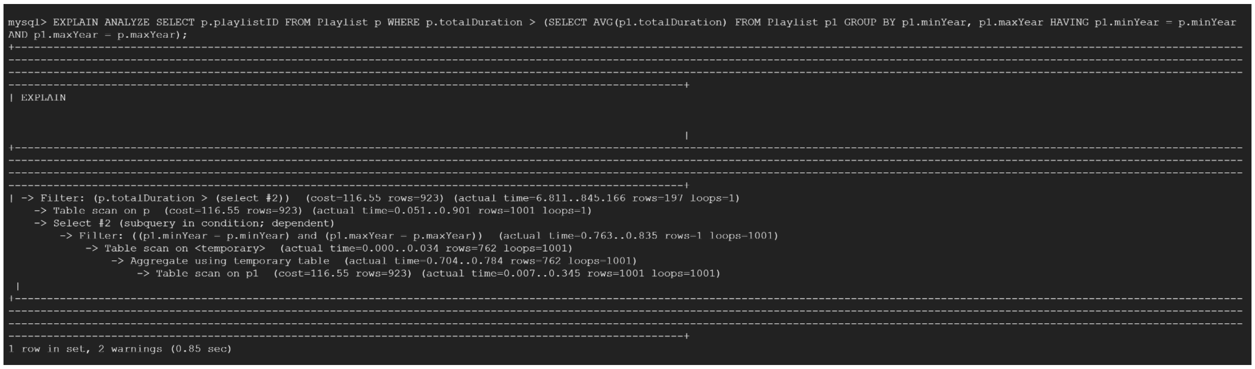
1) NO custom index



```
mysql> EXPLAIN ANALYZE SELECT p.playlistID
    -> FROM Playlist p
    -> WHERE p.totalDuration > (SELECT AVG(p1.totalDuration) FROM Playlist p1 GROUP BY p1.minYear, p1.maxYear HAVING p1.minYear = p.minYear AND p1.maxYear = p.maxYear);
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                              |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------+
| -> Filter: (p.totalDuration > (select #2))  (cost=116.55 rows=923) (actual time=6.929..836.408 rows=197 loops=1)
    -> Table scan on p  (cost=116.55 rows=923) (actual time=0.055..0.883 rows=1001 loops=1)
    -> Select #2 (subquery in condition; dependent)
        -> Filter: ((p1.minYear = p.minYear) and (p1.maxYear = p.maxYear))  (actual time=0.755..0.826 rows=1 loops=1001)
            -> Table scan on <temporary>  (actual time=0.000..0.034 rows=762 loops=1001)
                -> Aggregate using temporary table  (actual time=0.696..0.775 rows=762 loops=1001)
                    -> Table scan on p1  (cost=116.55 rows=923) (actual time=0.006..0.339 rows=1001 loops=1001)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------+
1 row in set, 2 warnings (0.84 sec)
```

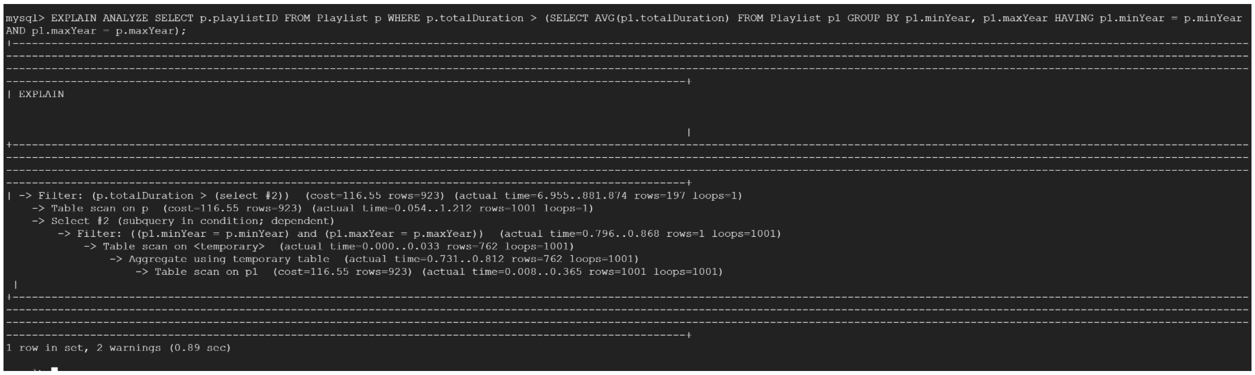
2) CREATE INDEX idx_dur ON Playlist (totalDuration):

Our first index that we attempted was on totalDuration. Filtering the duration improves, but the overall performance is minutely worse than the original index. The duration is used while filtering using Where so we see an improvement in performance. However, because this happens within a subquery and utilizing temporary tables, it does not make the biggest difference overall.



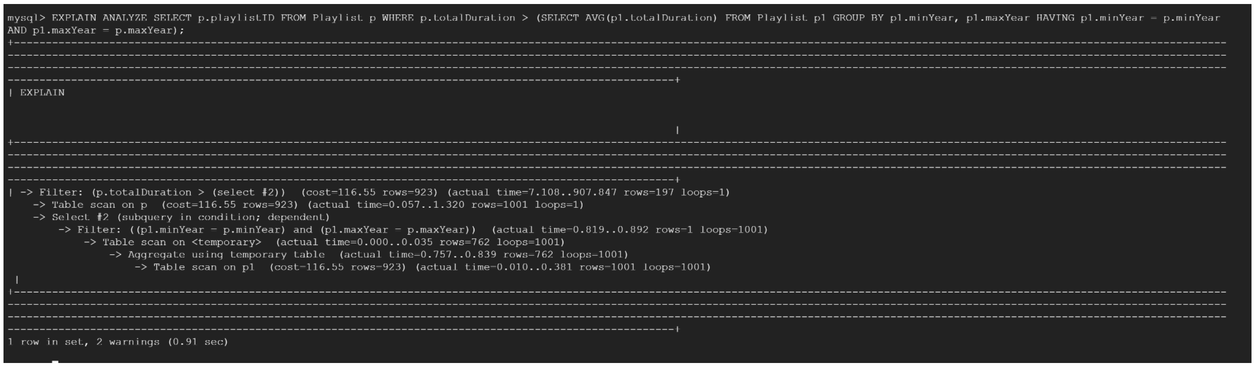
```
mysql> EXPLAIN ANALYZE SELECT p.playlistID FROM Playlist p WHERE p.totalDuration > (SELECT AVG(p1.totalDuration) FROM Playlist p1 GROUP BY p1.minYear, p1.maxYear HAVING p1.minYear = p.minYear
AND p1.maxYear = p.maxYear);
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                              |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------+
| -> Filter: (p.totalDuration > (select #2))  (cost=116.55 rows=923) (actual time=6.811..845.166 rows=197 loops=1)
    -> Table scan on p  (cost=116.55 rows=923) (actual time=0.051..0.901 rows=1001 loops=1)
    -> Select #2 (subquery in condition; dependent)
        -> Filter: ((p1.minYear = p.minYear) and (p1.maxYear = p.maxYear))  (actual time=0.763..0.835 rows=1 loops=1001)
            -> Table scan on <temporary>  (actual time=0.000..0.034 rows=762 loops=1001)
                -> Aggregate using temporary table  (actual time=0.704..0.784 rows=762 loops=1001)
                    -> Table scan on p1  (cost=116.55 rows=923) (actual time=0.007..0.345 rows=1001 loops=1001)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------+
1 row in set, 2 warnings (0.85 sec)
```

3) CREATE INDEX idx_min ON Playlist (minYear):
   We used minYear as an index because we use the min Year for comparison in the where clause. It has a performance worse than the original index. minYear is used in the group by and filtering for groups. We believe that this comparison is an easy comparison given logical operations and that making this the index slows the query.



```
mysql> EXPLAIN ANALYZE SELECT p.playlistID FROM Playlist p WHERE p.totalDuration > (SELECT AVG(p1.totalDuration) FROM Playlist p1 GROUP BY p1.minYear, p1.maxYear HAVING p1.minYear = p.minYear
AND p1.maxYear = p.maxYear);
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                                     |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
| -> Filter: (p.totalDuration > (select #2))  (cost=116.55 rows=923) (actual time=6.955..881.874 rows=197 loops=1)
    -> Table scan on p  (cost=116.55 rows=923) (actual time=0.054..1.212 rows=1001 loops=1)
    -> Select #2 (subquery in condition; dependent)
        -> Filter: ((p1.minYear = p.minYear) and (p1.maxYear = p.maxYear))  (actual time=0.796..0.868 rows=1 loops=1001)
            -> Table scan on <temporary>  (actual time=0.000..0.033 rows=762 loops=1001)
                -> Aggregate using temporary table  (actual time=0.731..0.812 rows=762 loops=1001)
                    -> Table scan on p1  (cost=116.55 rows=923) (actual time=0.008..0.365 rows=1001 loops=1001)
 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
1 row in set, 2 warnings (0.89 sec)
```

4) CREATE INDEX idx_max ON Playlist (maxYear):
   We used maxYear as an index because we use the max Year for comparison in the where clause. It has a performance worse than the original index. minYear is used in the group by and filtering for groups. We believe that this comparison is an easy comparison given logical operations and that making this the index slows the query. The usage of indices on minYear or maxYear just overcomplicates the processes.



```
mysql> EXPLAIN ANALYZE SELECT p.playlistID FROM Playlist p WHERE p.totalDuration > (SELECT AVG(p1.totalDuration) FROM Playlist p1 GROUP BY p1.minYear, p1.maxYear HAVING p1.minYear = p.minYear
AND p1.maxYear = p.maxYear);
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                                     |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
| -> Filter: (p.totalDuration > (select #2))  (cost=116.55 rows=923) (actual time=7.108..907.847 rows=197 loops=1)
    -> Table scan on p  (cost=116.55 rows=923) (actual time=0.057..1.320 rows=1001 loops=1)
    -> Select #2 (subquery in condition; dependent)
        -> Filter: ((p1.minYear = p.minYear) and (p1.maxYear = p.maxYear))  (actual time=0.819..0.892 rows=1 loops=1001)
            -> Table scan on <temporary>  (actual time=0.000..0.035 rows=762 loops=1001)
                -> Aggregate using temporary table  (actual time=0.757..0.839 rows=762 loops=1001)
                    -> Table scan on p1  (cost=116.55 rows=923) (actual time=0.010..0.381 rows=1001 loops=1001)
 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
1 row in set, 2 warnings (0.91 sec)
```

Overall: The best indexing option for this particular query is the PRIMARY default index, as all the custom indexes made the runtime longer than the original run.