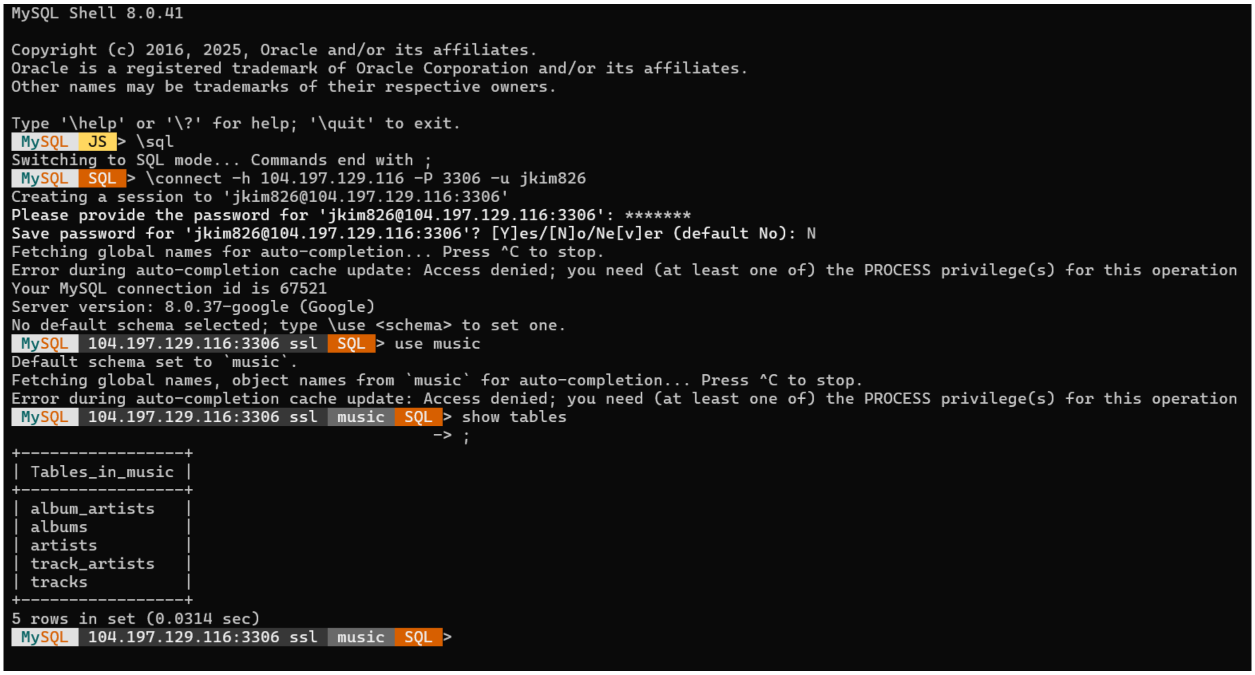
Part 1

Database on GCP Image



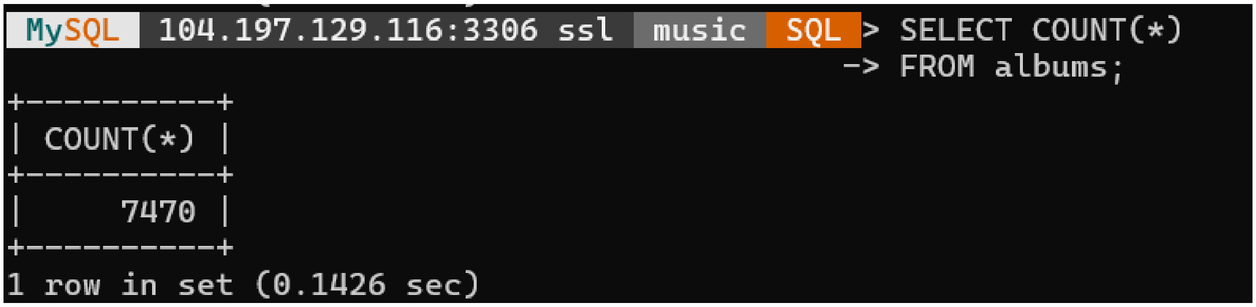
```
MySQL Shell 8.0.41

Copyright (c) 2016, 2025, 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 '\?' for help; '\quit' to exit.
 MySQL  JS > \sql
Switching to SQL mode... Commands end with ;
 MySQL  SQL > \connect -h 104.197.129.116 -P 3306 -u jkim826
Creating a session to 'jkim826@104.197.129.116:3306'
Please provide the password for 'jkim826@104.197.129.116:3306': *******
Save password for 'jkim826@104.197.129.116:3306'? [Y]es/[N]o/Ne[v]er (default No): N
Fetching global names for auto-completion... Press ^C to stop.
Error during auto-completion cache update: Access denied; you need (at least one of) the PROCESS privilege(s) for this operation
Your MySQL connection id is 67521
Server version: 8.0.37-google (Google)
No default schema selected; type \use <schema> to set one.
 MySQL  104.197.129.116:3306 ssl  SQL > use music
Default schema set to `music`.
Fetching global names, object names from `music` for auto-completion... Press ^C to stop.
Error during auto-completion cache update: Access denied; you need (at least one of) the PROCESS privilege(s) for this operation
 MySQL  104.197.129.116:3306 ssl  music  SQL > show tables
                                           -> ;
+-----------------+
| Tables_in_music |
+-----------------+
| album_artists   |
| albums          |
| artists         |
| track_artists   |
| tracks          |
+-----------------+
5 rows in set (0.0314 sec)
 MySQL  104.197.129.116:3306 ssl  music  SQL >
```

DDL Commands to Create Tables

- CREATE TABLE Album(AlbumId:INT [PK], AlbumPopularity:INT, AlbumName:VARCHAR(1000), TotalTracks:INT, AlbumDuration:INT)
- CREATE TABLE Artists(ArtistId:INT [PK], ArtistPopularity:INT, ArtistName:VARCHAR(1000))
- CREATE TABLE Songs (SongId:INT [PK], AlbumId:INT [FK], ArtistId:INT [FK], SongId:VARCHAR(1000), Explicit:INT, Energy:FLOAT, Danceability:FLOAT, Duration:INT)
- CREATE TABLE Streams(SongId:INT [PK][FK], Ranking:INT, NumStreams:INT)
- CREATE TABLE User(Username:VARCHAR(1000) [PK], Password:VARCHAR(1000), Email:VARCHAR(1000))
- CREATE TABLE UserTables(TableName:VARCHAR(1000) [PK], Username:VARCHAR(1000) [FK])

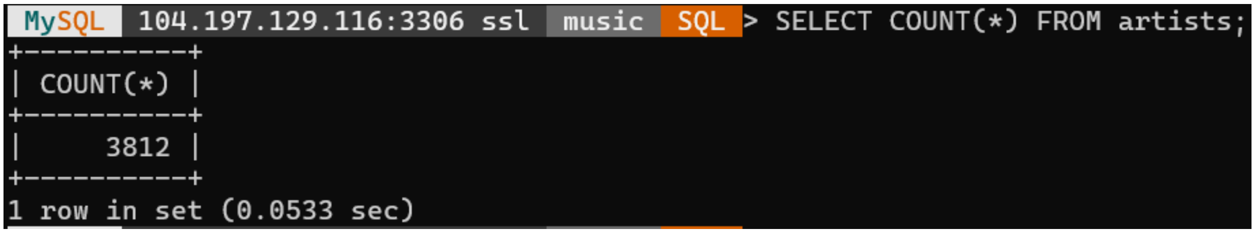
Images Showing At Least 1000 Rows in Three Different Tables Each

Albums Table:



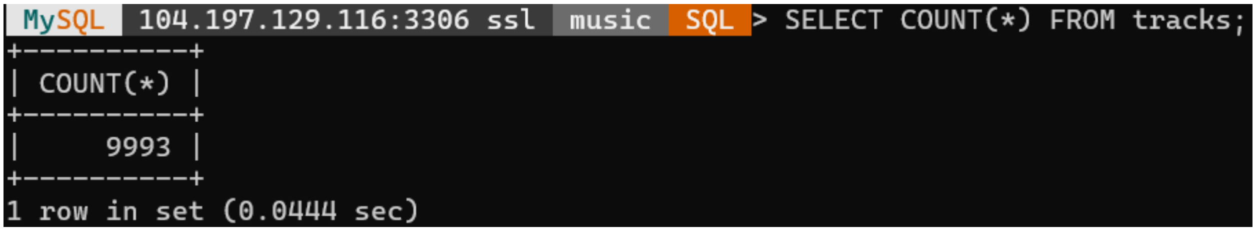
```
 MySQL  104.197.129.116:3306 ssl  music  SQL > SELECT COUNT(*)
                                            -> FROM albums;
+----------+
| COUNT(*) |
+----------+
|     7470 |
+----------+
1 row in set (0.1426 sec)
```

Artists Table:



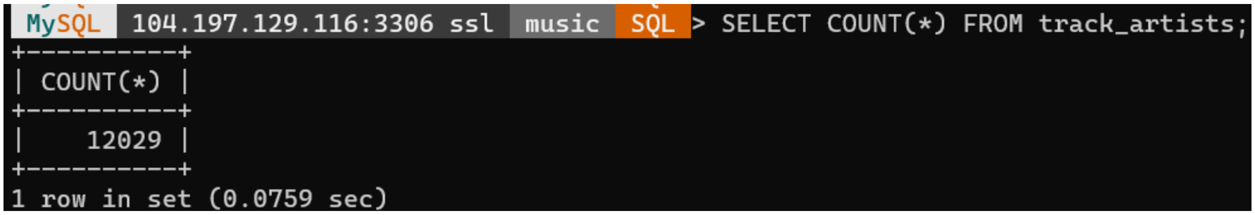
```
 MySQL  104.197.129.116:3306 ssl  music  SQL > SELECT COUNT(*) FROM artists;
+----------+
| COUNT(*) |
+----------+
|     3812 |
+----------+
1 row in set (0.0533 sec)
```

Tracks Table:



```
 MySQL  104.197.129.116:3306 ssl  music  SQL > SELECT COUNT(*) FROM tracks;
+----------+
| COUNT(*) |
+----------+
|     9993 |
+----------+
1 row in set (0.0444 sec)
```

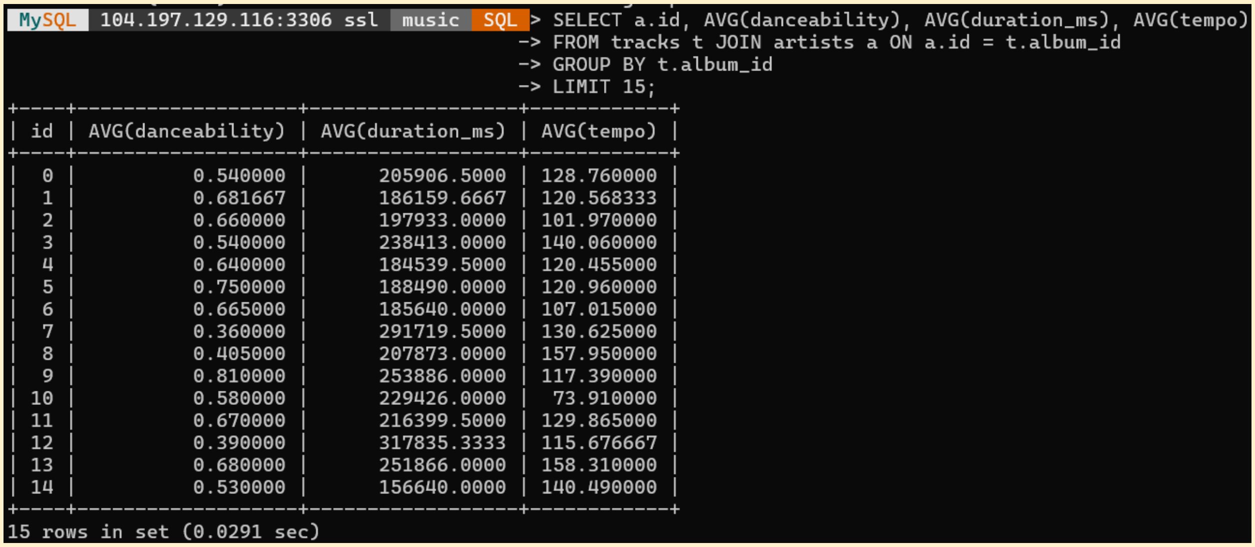
Track_artists Table:



```
 MySQL  104.197.129.116:3306 ssl  music  SQL > SELECT COUNT(*) FROM track_artists;
+----------+
| COUNT(*) |
+----------+
|    12029 |
+----------+
1 row in set (0.0759 sec)
```

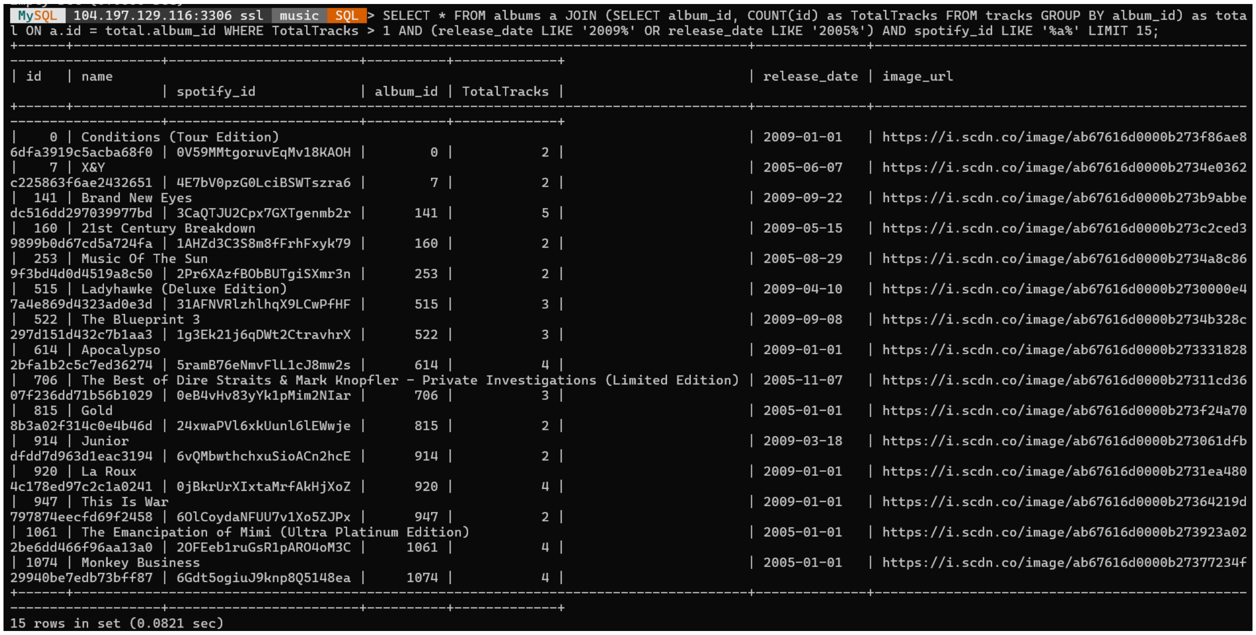
Advanced Queries & Top 15 Rows

- SELECT a.id, AVG(danceability), AVG(duration_ms), AVG(tempo) FROM tracks t JOIN artists a ON a.id = t.album_id GROUP BY t.album_id;



```
 MySQL  104.197.129.116:3306 ssl  music  SQL > SELECT a.id, AVG(danceability), AVG(duration_ms), AVG(tempo)
                                            -> FROM tracks t JOIN artists a ON a.id = t.album_id
                                            -> GROUP BY t.album_id
                                            -> LIMIT 15;
+----+-------------------+------------------+------------+
| id | AVG(danceability) | AVG(duration_ms) | AVG(tempo) |
+----+-------------------+------------------+------------+
|  0 |          0.540000 |      205906.5000 | 128.760000 |
|  1 |          0.681667 |      186159.6667 | 120.568333 |
|  2 |          0.660000 |      197933.0000 | 101.970000 |
|  3 |          0.540000 |      238413.0000 | 140.060000 |
|  4 |          0.640000 |      184539.5000 | 120.455000 |
|  5 |          0.750000 |      188490.0000 | 120.960000 |
|  6 |          0.665000 |      185640.0000 | 107.015000 |
|  7 |          0.360000 |      291719.5000 | 130.625000 |
|  8 |          0.405000 |      207873.0000 | 157.950000 |
|  9 |          0.810000 |      253886.0000 | 117.390000 |
| 10 |          0.580000 |      229426.0000 |  73.910000 |
| 11 |          0.670000 |      216399.5000 | 129.865000 |
| 12 |          0.390000 |      317835.3333 | 115.676667 |
| 13 |          0.680000 |      251866.0000 | 158.310000 |
| 14 |          0.530000 |      156640.0000 | 140.490000 |
+----+-------------------+------------------+------------+
15 rows in set (0.0291 sec)
```

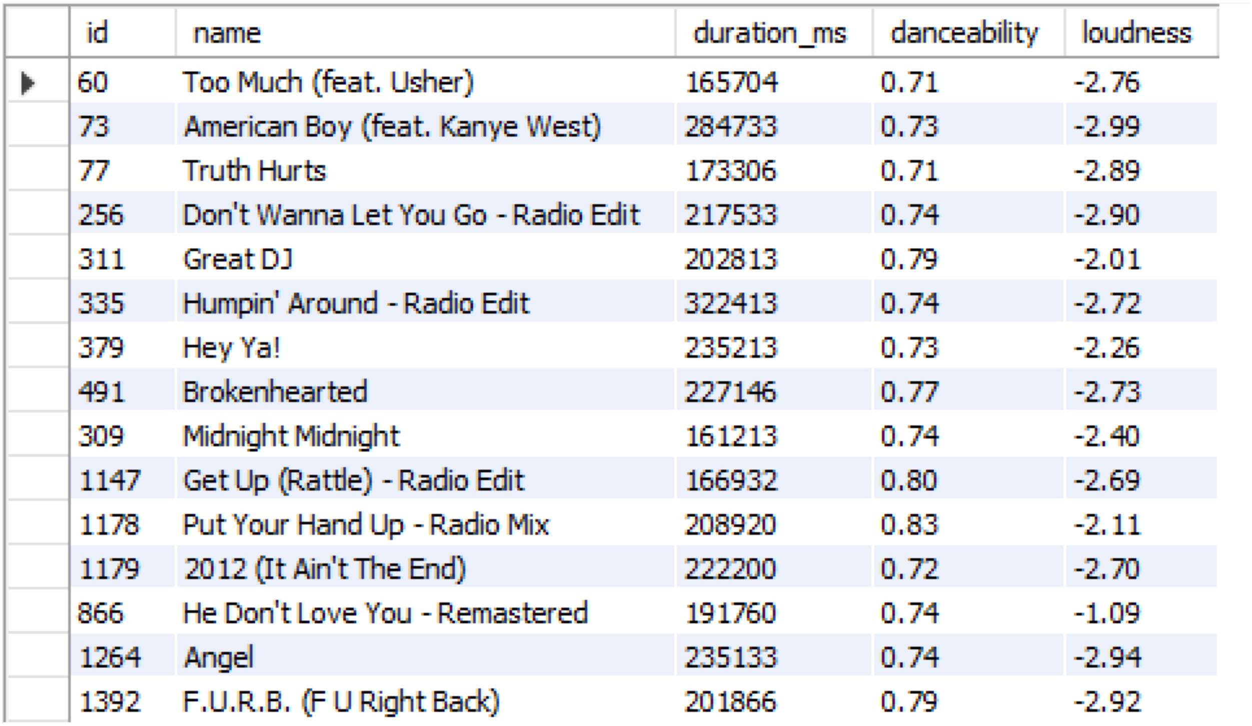
- SELECT * FROM albums a JOIN (SELECT album_id, COUNT(id) as TotalTracks FROM tracks GROUP BY album_id) as total ON a.id = total.album_id WHERE TotalTracks > 1 AND (release_date LIKE ‘2009%’ OR release_date LIKE ‘2005%’) AND spotify_id LIKE ‘%a%’;



```
 MySQL  104.197.129.116:3306 ssl  music  SQL > SELECT * FROM albums a JOIN (SELECT album_id, COUNT(id) as TotalTracks FROM tracks GROUP BY album_id) as tota
l ON a.id = total.album_id WHERE TotalTracks > 1 AND (release_date LIKE '2009%' OR release_date LIKE '2005%') AND spotify_id LIKE '%a%' LIMIT 15;
+------+---------------------------------------------------------------------------------------------+--------------+-----------------------------------------------
-------------------+------------------------+----------+-------------+
| id   | name                                                                                        | release_date | image_url
                   | spotify_id             | album_id | TotalTracks |
+------+---------------------------------------------------------------------------------------------+--------------+-----------------------------------------------
-------------------+------------------------+----------+-------------+
|    0 | Conditions (Tour Edition)                                                                   | 2009-01-01   | https://i.scdn.co/image/ab67616d0000b273f86ae8
6dfa3919c5acba68f0 | 0V59MMtgoruvEqMv18KAOH |        0 |           2 |
|    7 | X&Y                                                                                         | 2005-06-07   | https://i.scdn.co/image/ab67616d0000b2734e0362
c225863f6ae2432651 | 4E7bV0pzG0LciBSWTszra6 |        7 |           2 |
|  141 | Brand New Eyes                                                                              | 2009-09-22   | https://i.scdn.co/image/ab67616d0000b273b9abbe
dc516dd297039977bd | 3CaQTJU2Cpx7GXTgenmb2r |      141 |           5 |
|  160 | 21st Century Breakdown                                                                      | 2009-05-15   | https://i.scdn.co/image/ab67616d0000b273c2ced3
9899b0d67cd5a724fa | 1AHZd3C3S8m8fFrhFxyk79 |      160 |           2 |
|  253 | Music Of The Sun                                                                            | 2005-08-29   | https://i.scdn.co/image/ab67616d0000b2734a8c86
9f3bd4d0d4519a8c50 | 2Pr6XAzfBObBUTgiSXmr3n |      253 |           2 |
|  515 | Ladyhawke (Deluxe Edition)                                                                  | 2009-04-10   | https://i.scdn.co/image/ab67616d0000b2730000e4
7a4e869d4323ad0e3d | 31AFNVRlzhlhqX9LCwPfHF |      515 |           3 |
|  522 | The Blueprint 3                                                                             | 2009-09-08   | https://i.scdn.co/image/ab67616d0000b2734b328c
297d151d432c7b1aa3 | 1g3Ek21j6qDWt2CtravhrX |      522 |           3 |
|  614 | Apocalypso                                                                                  | 2009-01-01   | https://i.scdn.co/image/ab67616d0000b273331828
2bfa1b2c5c7ed36274 | 5ramB76eNmvFlL1cJ8mw2s |      614 |           4 |
|  706 | The Best of Dire Straits & Mark Knopfler - Private Investigations (Limited Edition)         | 2005-11-07   | https://i.scdn.co/image/ab67616d0000b27311cd36
07f236dd71b56b1029 | 0eB4vHv83yYk1pMim2NIar |      706 |           3 |
|  815 | Gold                                                                                        | 2005-01-01   | https://i.scdn.co/image/ab67616d0000b273f24a70
8b3a02f314c0e4b46d | 24xwaPVl6xkUunl6lEWwje |      815 |           2 |
|  914 | Junior                                                                                      | 2009-03-18   | https://i.scdn.co/image/ab67616d0000b273061dfb
dfdd7d963d1eac3194 | 6vQMbwthchxuSioACn2hcE |      914 |           2 |
|  920 | La Roux                                                                                     | 2009-01-01   | https://i.scdn.co/image/ab67616d0000b2731ea480
4c178ed97c2c1a0241 | 0jBkrUrXIxtaMrfAkHjXoZ |      920 |           4 |
|  947 | This Is War                                                                                 | 2009-01-01   | https://i.scdn.co/image/ab67616d0000b27364219d
797874eecfd69f2458 | 6OlCoydaNFUU7v1Xo5ZJPx |      947 |           2 |
| 1061 | The Emancipation of Mimi (Ultra Platinum Edition)                                           | 2005-01-01   | https://i.scdn.co/image/ab67616d0000b273923a02
2be6dd466f96aa13a0 | 2OFEeb1ruGsR1pARO4oM3C |     1061 |           4 |
| 1074 | Monkey Business                                                                             | 2005-01-01   | https://i.scdn.co/image/ab67616d0000b27377234f
29940be7edb73bff87 | 6Gdt5ogiuJ9knp8Q5148ea |     1074 |           4 |
+------+---------------------------------------------------------------------------------------------+--------------+-----------------------------------------------
-------------------+------------------------+----------+-------------+
15 rows in set (0.0821 sec)
```

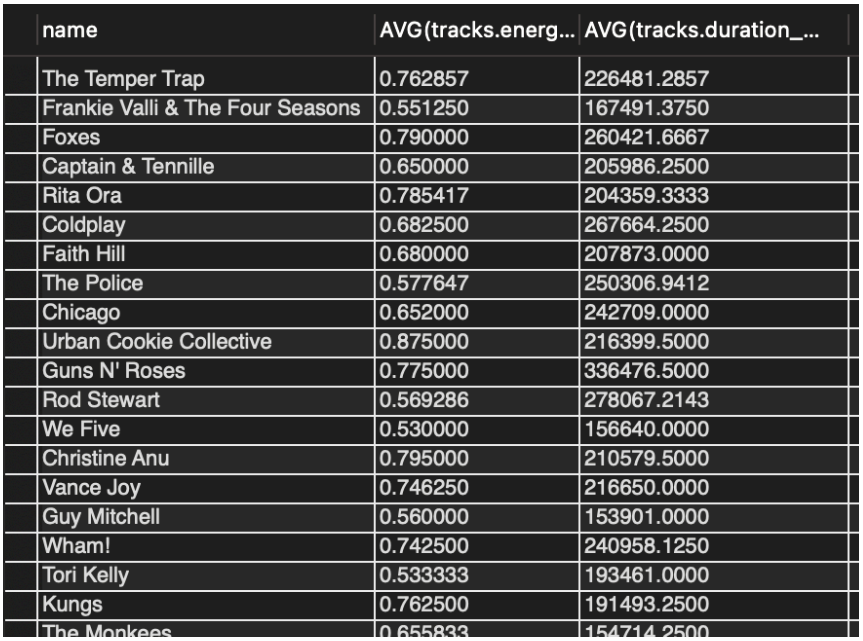
- SELECT a.id, t.name, t.duration_ms, t.danceability, t.loudness FROM albums a JOIN tracks t ON a.id = t.album_id WHERE t.duration_ms > 23000 and danceability > 0.70

and loudness > -3 UNION SELECT a2.id, t2.name, t2.duration_ms, t2.danceability, t2.loudness FROM albums a2 JOIN tracks t2 on a2.id = t2.album_id WHERE t2.duration_ms < 20000 and t2.danceability < 0.30 and t2.loudness < -8;



id	name	duration_ms	danceability	loudness
60	Too Much (feat. Usher)	165704	0.71	-2.76
73	American Boy (feat. Kanye West)	284733	0.73	-2.99
77	Truth Hurts	173306	0.71	-2.89
256	Don't Wanna Let You Go - Radio Edit	217533	0.74	-2.90
311	Great DJ	202813	0.79	-2.01
335	Humpin' Around - Radio Edit	322413	0.74	-2.72
379	Hey Ya!	235213	0.73	-2.26
491	Brokenhearted	227146	0.77	-2.73
309	Midnight Midnight	161213	0.74	-2.40
1147	Get Up (Rattle) - Radio Edit	166932	0.80	-2.69
1178	Put Your Hand Up - Radio Mix	208920	0.83	-2.11
1179	2012 (It Ain't The End)	222200	0.72	-2.70
866	He Don't Love You - Remastered	191760	0.74	-1.09
1264	Angel	235133	0.74	-2.94
1392	F.U.R.B. (F U Right Back)	201866	0.79	-2.92

- SELECT artists.name, AVG(tracks.energy), AVG(tracks.duration_ms) FROM (tracks JOIN track_artists ON tracks.id = track_artists.track_id) JOIN artists ON track_artists.artist_id = artists.id GROUP BY artists.name

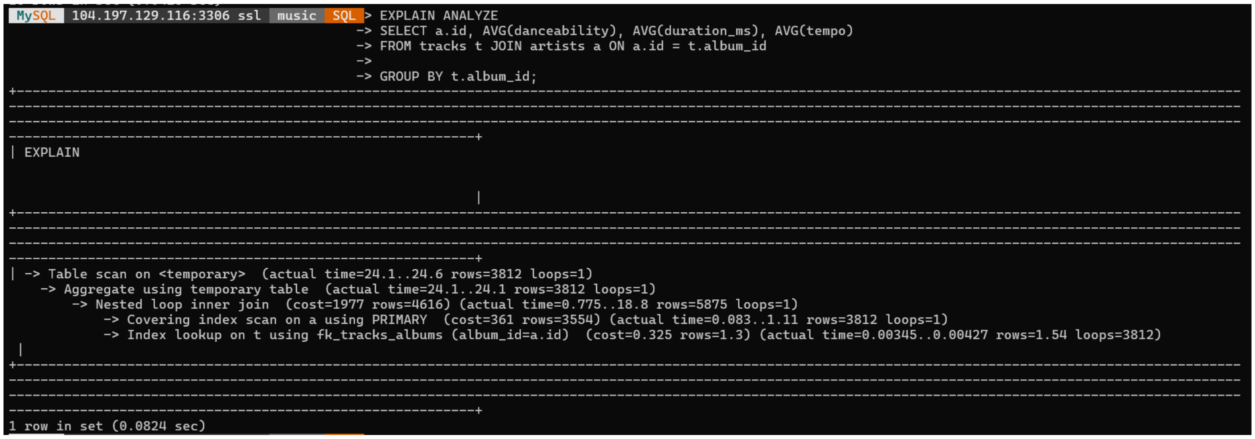


name	AVG(tracks.energ...	AVG(tracks.duration_...
The Temper Trap	0.762857	226481.2857
Frankie Valli & The Four Seasons	0.551250	167491.3750
Foxes	0.790000	260421.6667
Captain & Tennille	0.650000	205986.2500
Rita Ora	0.785417	204359.3333
Coldplay	0.682500	267664.2500
Faith Hill	0.680000	207873.0000
The Police	0.577647	250306.9412
Chicago	0.652000	242709.0000
Urban Cookie Collective	0.875000	216399.5000
Guns N' Roses	0.775000	336476.5000
Rod Stewart	0.569286	278067.2143
We Five	0.530000	156640.0000
Christine Anu	0.795000	210579.5000
Vance Joy	0.746250	216650.0000
Guy Mitchell	0.560000	153901.0000
Wham!	0.742500	240958.1250
Tori Kelly	0.533333	193461.0000
Kungs	0.762500	191493.2500
The Monkees	0.655833	154714.2500

Part 2: Indexing

Command 1: SELECT a.id, AVG(danceability), AVG(duration_ms), AVG(tempo) FROM tracks t JOIN artists a ON a.id = t.album_id GROUP BY t.album_id;

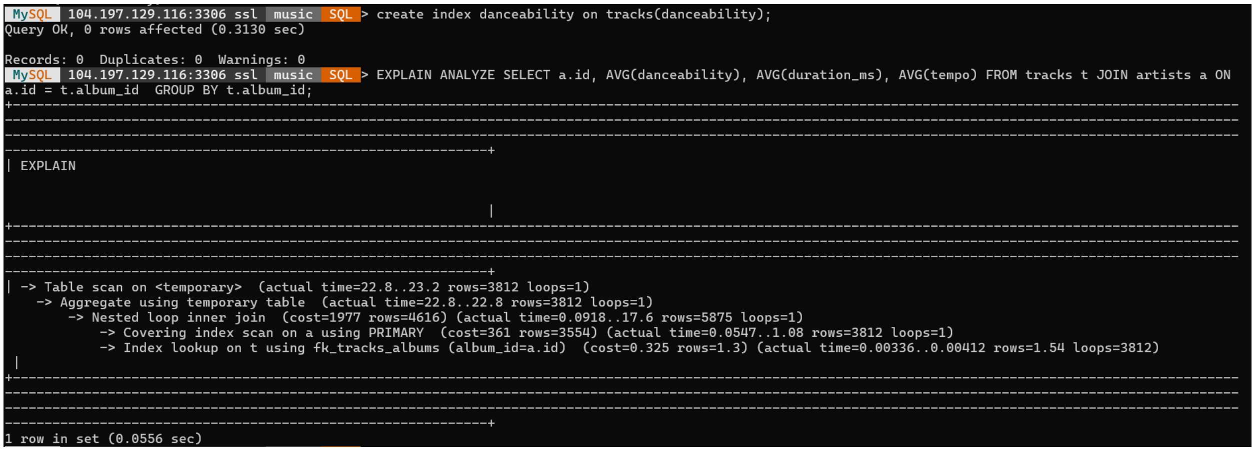
Initial EXPLAIN ANALYZE w/ No Indexing:



```
MySQL  104.197.129.116:3306 ssl  music  SQL > EXPLAIN ANALYZE
                                           -> SELECT a.id, AVG(danceability), AVG(duration_ms), AVG(tempo)
                                           -> FROM tracks t JOIN artists a ON a.id = t.album_id
                                           ->
                                           -> GROUP BY t.album_id;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------+
| EXPLAIN


                                                            |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=24.1..24.6 rows=3812 loops=1)
    -> Aggregate using temporary table  (actual time=24.1..24.1 rows=3812 loops=1)
        -> Nested loop inner join  (cost=1977 rows=4616) (actual time=0.775..18.8 rows=5875 loops=1)
            -> Covering index scan on a using PRIMARY  (cost=361 rows=3554) (actual time=0.083..1.11 rows=3812 loops=1)
            -> Index lookup on t using fk_tracks_albums (album_id=a.id)  (cost=0.325 rows=1.3) (actual time=0.00345..0.00427 rows=1.54 loops=3812)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------+
1 row in set (0.0824 sec)
```

- The cost is 1977 without Indexing.

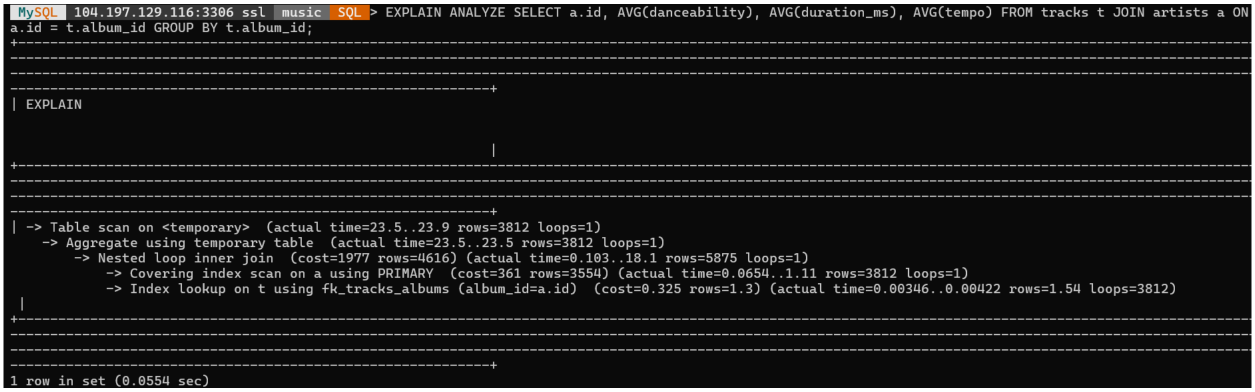
EXPLAIN ANALYZE with index on tracks(danceability):



```
MySQL  104.197.129.116:3306 ssl  music  SQL > create index danceability on tracks(danceability);
Query OK, 0 rows affected (0.3130 sec)

Records: 0  Duplicates: 0  Warnings: 0
MySQL  104.197.129.116:3306 ssl  music  SQL > EXPLAIN ANALYZE SELECT a.id, AVG(danceability), AVG(duration_ms), AVG(tempo) FROM tracks t JOIN artists a ON
a.id = t.album_id  GROUP BY t.album_id;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
| EXPLAIN


                                                             |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=22.8..23.2 rows=3812 loops=1)
    -> Aggregate using temporary table  (actual time=22.8..22.8 rows=3812 loops=1)
        -> Nested loop inner join  (cost=1977 rows=4616) (actual time=0.0918..17.6 rows=5875 loops=1)
            -> Covering index scan on a using PRIMARY  (cost=361 rows=3554) (actual time=0.0547..1.08 rows=3812 loops=1)
            -> Index lookup on t using fk_tracks_albums (album_id=a.id)  (cost=0.325 rows=1.3) (actual time=0.00336..0.00412 rows=1.54 loops=3812)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
1 row in set (0.0556 sec)
```

- The cost is 1977 with this index.

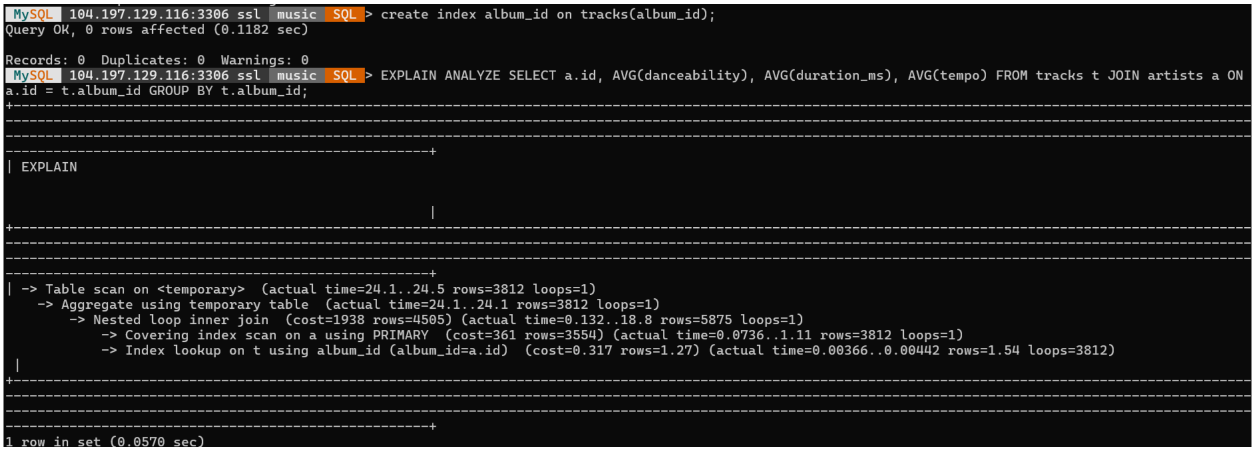
EXPLAIN ANALYZE with index on tracks(duration_ms):



```
 MySQL  104.197.129.116:3306 ssl  music  SQL > EXPLAIN ANALYZE SELECT a.id, AVG(danceability), AVG(duration_ms), AVG(tempo) FROM tracks t JOIN artists a ON
a.id = t.album_id GROUP BY t.album_id;
+-------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------+
| EXPLAIN


                                                           |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=23.5..23.9 rows=3812 loops=1)
    -> Aggregate using temporary table  (actual time=23.5..23.5 rows=3812 loops=1)
        -> Nested loop inner join  (cost=1977 rows=4616) (actual time=0.103..18.1 rows=5875 loops=1)
            -> Covering index scan on a using PRIMARY  (cost=361 rows=3554) (actual time=0.0654..1.11 rows=3812 loops=1)
            -> Index lookup on t using fk_tracks_albums (album_id=a.id)  (cost=0.325 rows=1.3) (actual time=0.00346..0.00422 rows=1.54 loops=3812)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------+
1 row in set (0.0554 sec)
```

- The cost is 1977 with this index.

EXPLAIN ANALYZE with index on tracks(album_id):



```
 MySQL  104.197.129.116:3306 ssl  music  SQL > create index album_id on tracks(album_id);
Query OK, 0 rows affected (0.1182 sec)

Records: 0  Duplicates: 0  Warnings: 0
 MySQL  104.197.129.116:3306 ssl  music  SQL > EXPLAIN ANALYZE SELECT a.id, AVG(danceability), AVG(duration_ms), AVG(tempo) FROM tracks t JOIN artists a ON
a.id = t.album_id GROUP BY t.album_id;
+-------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------+
| EXPLAIN


                                                     |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------+
| -> Table scan on <temporary>  (actual time=24.1..24.5 rows=3812 loops=1)
    -> Aggregate using temporary table  (actual time=24.1..24.1 rows=3812 loops=1)
        -> Nested loop inner join  (cost=1938 rows=4505) (actual time=0.132..18.8 rows=5875 loops=1)
            -> Covering index scan on a using PRIMARY  (cost=361 rows=3554) (actual time=0.0736..1.11 rows=3812 loops=1)
            -> Index lookup on t using album_id (album_id=a.id)  (cost=0.317 rows=1.27) (actual time=0.00366..0.00442 rows=1.54 loops=3812)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------+
1 row in set (0.0570 sec)
```

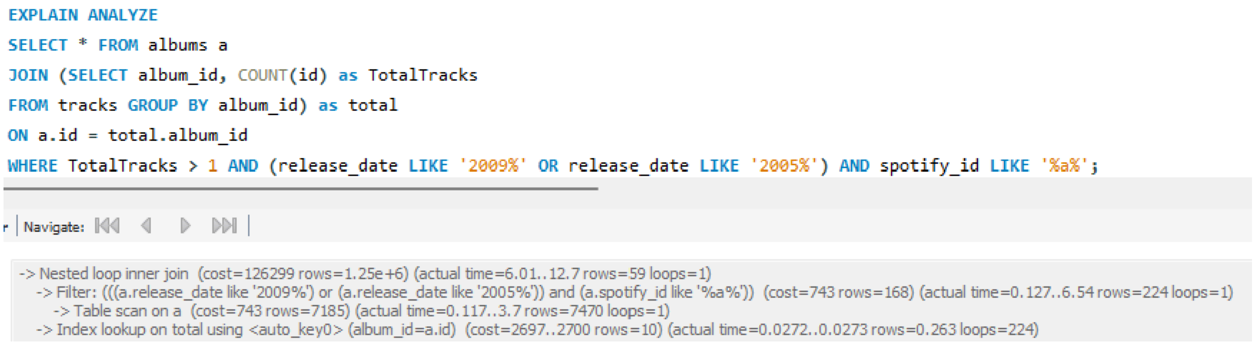
- The cost is 1938 with this index.

FINAL INDEX DESIGN:

We decided on a final index design of an index on tracks(album_id). The reason for this is that we can't index a primary key (a.id is a primary key in this query), and the other attributes in our query don't make an impact on reducing the cost of our query. We tried two different columns in the SELECT clauses, and indexing those columns made no impact on reducing the cost. We can see that our original query without indexing had a cost of 1977. Creating an index on tracks(danceability) and tracks(duration_ms) did not reduce the cost, as we can see from the first two screenshots. However, indexing tracks(album_id) did reduce the cost slightly (from 1977 to 1938). This is likely because the tracks(album_id) attribute appears in the JOIN clause, whereas the other attributes do not. Therefore, our group decided to use a final index design of just having an index on tracks(album_id).

Command 2: SELECT * FROM albums a JOIN (SELECT album_id, COUNT(id) as TotalTracks FROM tracks GROUP BY album_id) as total ON a.id = total.album_id WHERE TotalTracks > 1 AND (release_date LIKE ‘2009%’ OR release_date LIKE ‘2005%’) AND spotify_id LIKE ‘%a%’;

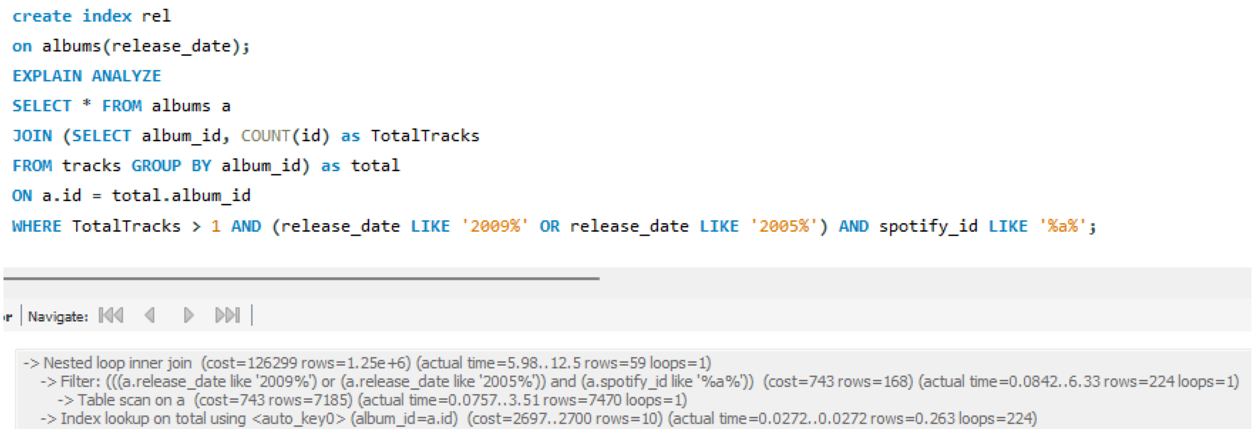
Initial EXPLAIN ANALYZE w/ No Indexing:



```
EXPLAIN ANALYZE
SELECT * FROM albums a
JOIN (SELECT album_id, COUNT(id) as TotalTracks
FROM tracks GROUP BY album_id) as total
ON a.id = total.album_id
WHERE TotalTracks > 1 AND (release_date LIKE '2009%' OR release_date LIKE '2005%') AND spotify_id LIKE '%a%';
```

```
-> Nested loop inner join  (cost=126299 rows=1.25e+6) (actual time=6.01..12.7 rows=59 loops=1)
    -> Filter: (((a.release_date like '2009%') or (a.release_date like '2005%')) and (a.spotify_id like '%a%'))  (cost=743 rows=168) (actual time=0.127..6.54 rows=224 loops=1)
        -> Table scan on a  (cost=743 rows=7185) (actual time=0.117..3.7 rows=7470 loops=1)
    -> Index lookup on total using <auto_key0> (album_id=a.id)  (cost=2697..2700 rows=10) (actual time=0.0272..0.0273 rows=0.263 loops=224)
```

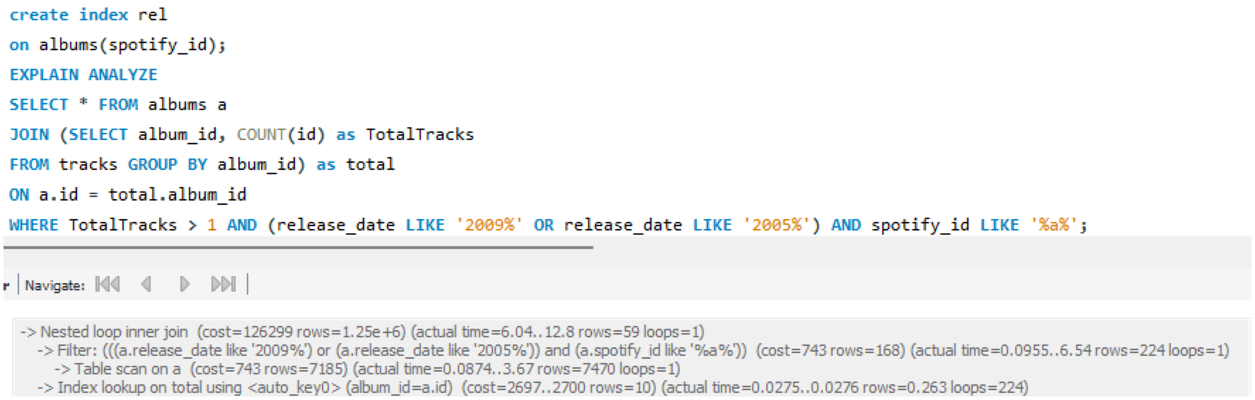
- **Cost: 126,999 w/o indexing**



```
create index rel
on albums(release_date);
EXPLAIN ANALYZE
SELECT * FROM albums a
JOIN (SELECT album_id, COUNT(id) as TotalTracks
FROM tracks GROUP BY album_id) as total
ON a.id = total.album_id
WHERE TotalTracks > 1 AND (release_date LIKE '2009%' OR release_date LIKE '2005%') AND spotify_id LIKE '%a%';
```

```
-> Nested loop inner join  (cost=126299 rows=1.25e+6) (actual time=5.98..12.5 rows=59 loops=1)
    -> Filter: (((a.release_date like '2009%') or (a.release_date like '2005%')) and (a.spotify_id like '%a%'))  (cost=743 rows=168) (actual time=0.0842..6.33 rows=224 loops=1)
        -> Table scan on a  (cost=743 rows=7185) (actual time=0.0757..3.51 rows=7470 loops=1)
    -> Index lookup on total using <auto_key0> (album_id=a.id)  (cost=2697..2700 rows=10) (actual time=0.0272..0.0272 rows=0.263 loops=224)
```

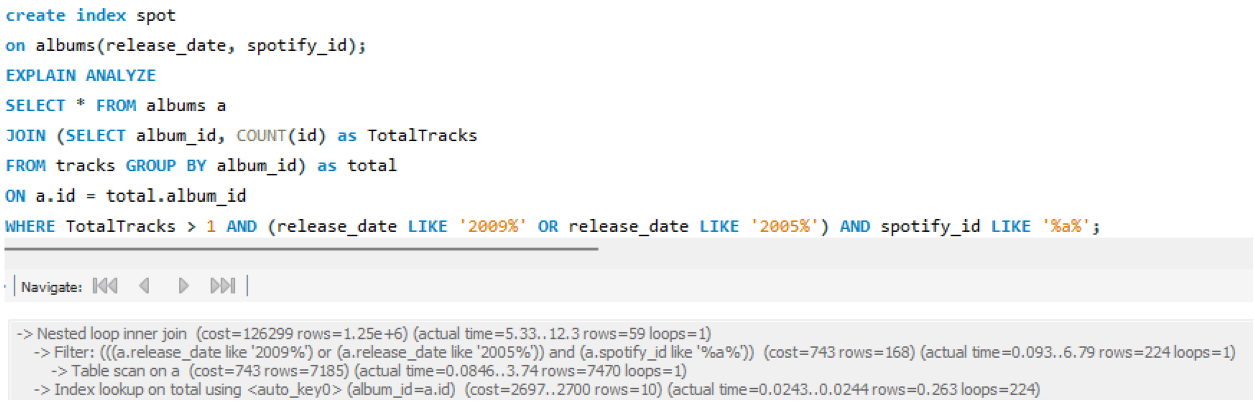
- **Cost: 126,999 w/ indexing on albums.release_date**



```
create index rel
on albums(spotify_id);
EXPLAIN ANALYZE
SELECT * FROM albums a
JOIN (SELECT album_id, COUNT(id) as TotalTracks
FROM tracks GROUP BY album_id) as total
ON a.id = total.album_id
WHERE TotalTracks > 1 AND (release_date LIKE '2009%' OR release_date LIKE '2005%') AND spotify_id LIKE '%a%';
```

```
-> Nested loop inner join  (cost=126299 rows=1.25e+6) (actual time=6.04..12.8 rows=59 loops=1)
    -> Filter: (((a.release_date like '2009%') or (a.release_date like '2005%')) and (a.spotify_id like '%a%'))  (cost=743 rows=168) (actual time=0.0955..6.54 rows=224 loops=1)
        -> Table scan on a  (cost=743 rows=7185) (actual time=0.0874..3.67 rows=7470 loops=1)
    -> Index lookup on total using <auto_key0> (album_id=a.id)  (cost=2697..2700 rows=10) (actual time=0.0275..0.0276 rows=0.263 loops=224)
```

- **Cost: 126,999 w/ indexing on albums.spotify_id**



```sql
create index spot
on albums(release_date, spotify_id);
EXPLAIN ANALYZE
SELECT * FROM albums a
JOIN (SELECT album_id, COUNT(id) as TotalTracks
FROM tracks GROUP BY album_id) as total
ON a.id = total.album_id
WHERE TotalTracks > 1 AND (release_date LIKE '2009%' OR release_date LIKE '2005%') AND spotify_id LIKE '%a%';
```

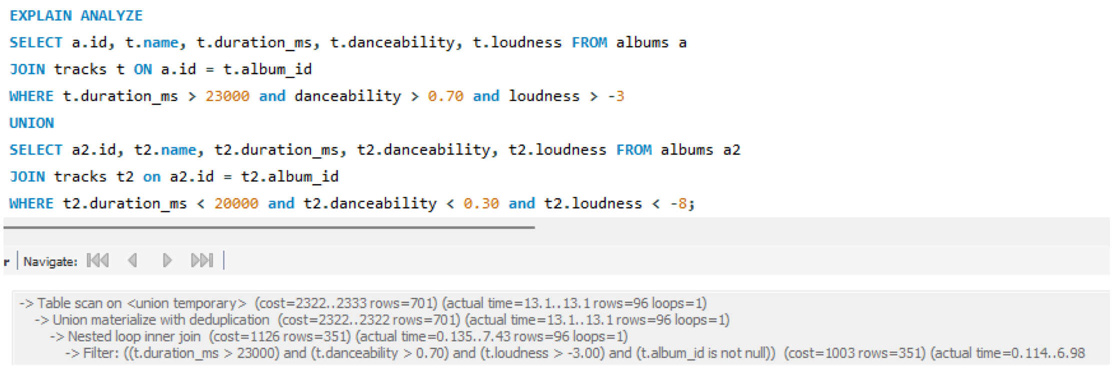
```
-> Nested loop inner join  (cost=126299 rows=1.25e+6) (actual time=5.33..12.3 rows=59 loops=1)
   -> Filter: (((a.release_date like '2009%') or (a.release_date like '2005%')) and (a.spotify_id like '%a%'))  (cost=743 rows=168) (actual time=0.093..6.79 rows=224 loops=1)
      -> Table scan on a  (cost=743 rows=7185) (actual time=0.0846..3.74 rows=7470 loops=1)
   -> Index lookup on total using <auto_key0> (album_id=a.id)  (cost=2697..2700 rows=10) (actual time=0.0243..0.0244 rows=0.263 loops=224)
```

- **Cost: 126,999 w/ indexing on albums.spotify_id and albums.release_date**

In this query, we were unable to reduce the cost in every implementation of indexing. This is likely due to the fact that the 'WHERE' condition includes the '%' symbol which represents a string "wildcard". As a result, the query must examine every single row in the database regardless of any indexing. For instance, in the condition "spotidy_id LIKE '%a%'", the database must scan all values to see which rows contain the character 'a'. Therefore, indexing did not improve the cost of our query.

Command 3: SELECT a.id, t.name, t.duration_ms, t.danceability, t.loudness FROM albums a JOIN tracks t ON a.id = t.album_id WHERE t.duration_ms > 23000 and danceability > 0.70 and loudness > -3 UNION SELECT a2.id, t2.name, t2.duration_ms, t2.danceability, t2.loudness FROM albums a2 JOIN tracks t2 on a2.id = t2.album_id WHERE t2.duration_ms < 20000 and t2.danceability < 0.30 and t2.loudness < -8;

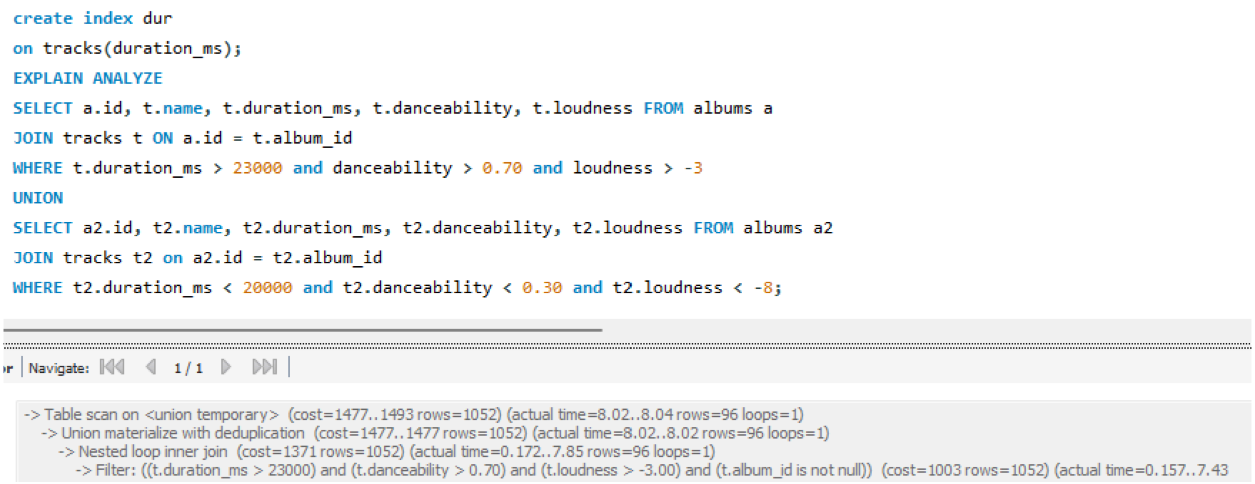
__EXPLAIN ANALYZE w/o index__



```sql
EXPLAIN ANALYZE
SELECT a.id, t.name, t.duration_ms, t.danceability, t.loudness FROM albums a
JOIN tracks t ON a.id = t.album_id
WHERE t.duration_ms > 23000 and danceability > 0.70 and loudness > -3
UNION
SELECT a2.id, t2.name, t2.duration_ms, t2.danceability, t2.loudness FROM albums a2
JOIN tracks t2 on a2.id = t2.album_id
WHERE t2.duration_ms < 20000 and t2.danceability < 0.30 and t2.loudness < -8;
```

```
-> Table scan on <union temporary>  (cost=2322..2333 rows=701) (actual time=13.1..13.1 rows=96 loops=1)
   -> Union materialize with deduplication  (cost=2322..2322 rows=701) (actual time=13.1..13.1 rows=96 loops=1)
      -> Nested loop inner join  (cost=1126 rows=351) (actual time=0.135..7.43 rows=96 loops=1)
         -> Filter: ((t.duration_ms > 23000) and (t.danceability > 0.70) and (t.loudness > -3.00) and (t.album_id is not null))  (cost=1003 rows=351) (actual time=0.114..6.98
```

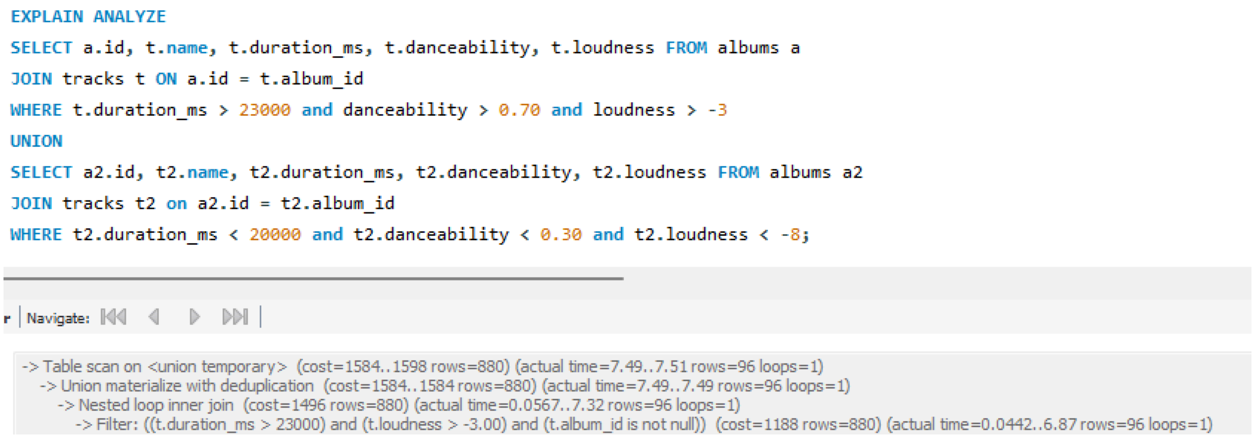
- **Cost: 2322 w/o indexing**



```sql
create index dur
on tracks(duration_ms);
EXPLAIN ANALYZE
SELECT a.id, t.name, t.duration_ms, t.danceability, t.loudness FROM albums a
JOIN tracks t ON a.id = t.album_id
WHERE t.duration_ms > 23000 and danceability > 0.70 and loudness > -3
UNION
SELECT a2.id, t2.name, t2.duration_ms, t2.danceability, t2.loudness FROM albums a2
JOIN tracks t2 on a2.id = t2.album_id
WHERE t2.duration_ms < 20000 and t2.danceability < 0.30 and t2.loudness < -8;
```

```
-> Table scan on <union temporary>  (cost=1477..1493 rows=1052) (actual time=8.02..8.04 rows=96 loops=1)
  -> Union materialize with deduplication  (cost=1477..1477 rows=1052) (actual time=8.02..8.02 rows=96 loops=1)
    -> Nested loop inner join  (cost=1371 rows=1052) (actual time=0.172..7.85 rows=96 loops=1)
      -> Filter: ((t.duration_ms > 23000) and (t.danceability > 0.70) and (t.loudness > -3.00) and (t.album_id is not null))  (cost=1003 rows=1052) (actual time=0.157..7.43
```

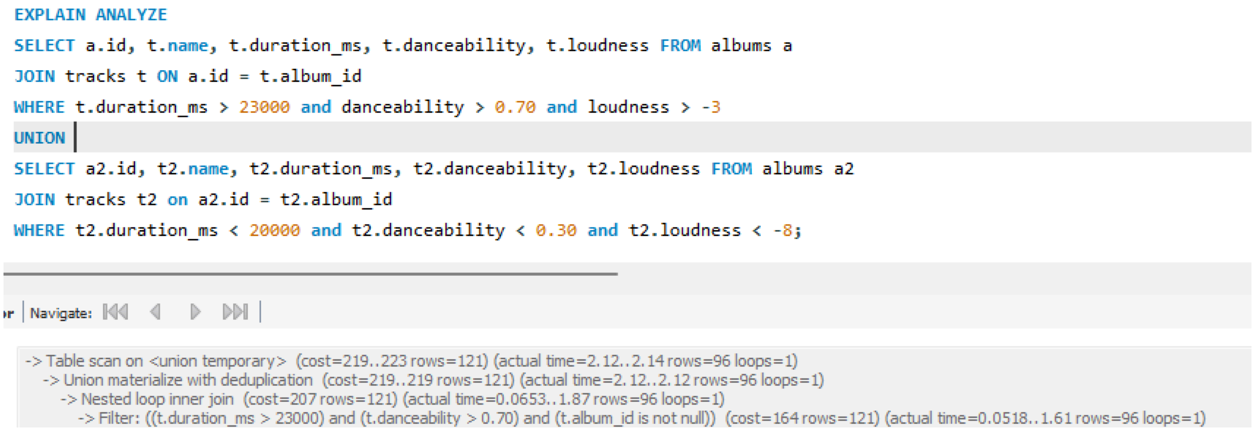
- **Cost: 1477 with indexing on tracks.duration_ms**



```sql
EXPLAIN ANALYZE
SELECT a.id, t.name, t.duration_ms, t.danceability, t.loudness FROM albums a
JOIN tracks t ON a.id = t.album_id
WHERE t.duration_ms > 23000 and danceability > 0.70 and loudness > -3
UNION
SELECT a2.id, t2.name, t2.duration_ms, t2.danceability, t2.loudness FROM albums a2
JOIN tracks t2 on a2.id = t2.album_id
WHERE t2.duration_ms < 20000 and t2.danceability < 0.30 and t2.loudness < -8;
```

```
-> Table scan on <union temporary>  (cost=1584..1598 rows=880) (actual time=7.49..7.51 rows=96 loops=1)
  -> Union materialize with deduplication  (cost=1584..1584 rows=880) (actual time=7.49..7.49 rows=96 loops=1)
    -> Nested loop inner join  (cost=1496 rows=880) (actual time=0.0567..7.32 rows=96 loops=1)
      -> Filter: ((t.duration_ms > 23000) and (t.loudness > -3.00) and (t.album_id is not null))  (cost=1188 rows=880) (actual time=0.0442..6.87 rows=96 loops=1)
```

- **Cost: 1584 with indexing on tracks.duration_ms and tracks.danceability**



```sql
EXPLAIN ANALYZE
SELECT a.id, t.name, t.duration_ms, t.danceability, t.loudness FROM albums a
JOIN tracks t ON a.id = t.album_id
WHERE t.duration_ms > 23000 and danceability > 0.70 and loudness > -3
UNION
SELECT a2.id, t2.name, t2.duration_ms, t2.danceability, t2.loudness FROM albums a2
JOIN tracks t2 on a2.id = t2.album_id
WHERE t2.duration_ms < 20000 and t2.danceability < 0.30 and t2.loudness < -8;
```

```
-> Table scan on <union temporary>  (cost=219..223 rows=121) (actual time=2.12..2.14 rows=96 loops=1)
  -> Union materialize with deduplication  (cost=219..219 rows=121) (actual time=2.12..2.12 rows=96 loops=1)
    -> Nested loop inner join  (cost=207 rows=121) (actual time=0.0653..1.87 rows=96 loops=1)
      -> Filter: ((t.duration_ms > 23000) and (t.danceability > 0.70) and (t.album_id is not null))  (cost=164 rows=121) (actual time=0.0518..1.61 rows=96 loops=1)
```

- **Cost: 219 with indexing on tracks.duration_ms and tracks.loudness**

Indexing Analysis:

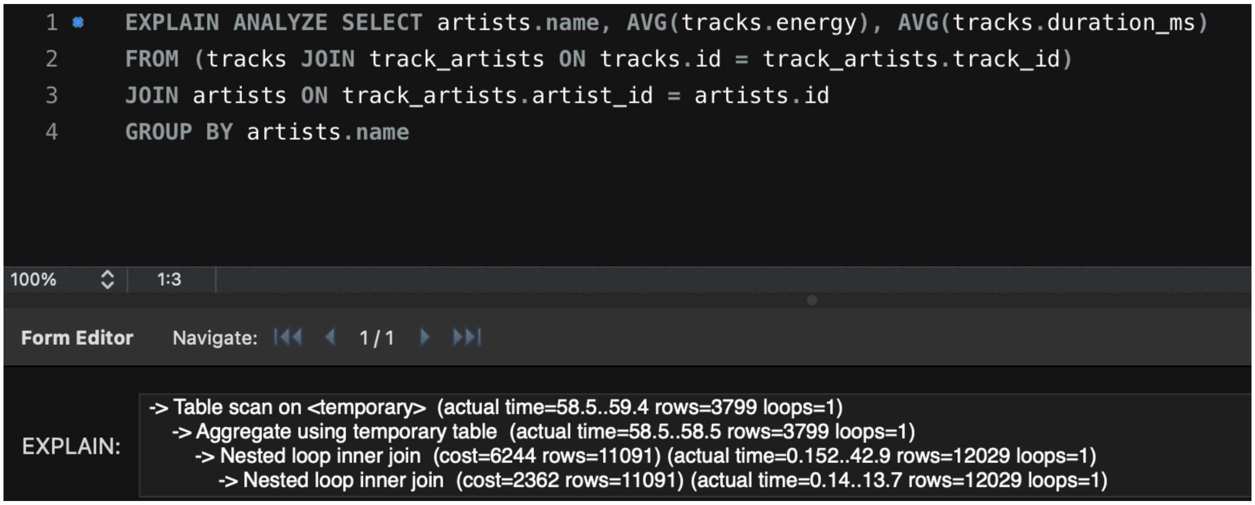
For the final indexing design for this query, we chose to go with indexing on two columns, tracks.duration_ms and tracks.loudness. When running the command with EXPLAIN ANALYZE, we achieved a cost of 219.

In our first implementation of indexing, we noticed that the cost dramatically decreased from the original query (w/o indexing) and this made sense since it helped the query have better lookup times when filtering out rows. So, in an attempt to further reduce the query search time, for our second implementation of indexing, we indexed two columns: tracks.duration_ms and tracks.danceability. However, this implementation only increased the cost compared to a single indexing of tracks.duration_ms. One reason this might be is because this format of indexing reduced the index's selectivity, thus causing the query to examine more rows than the prior.

Finally, for our third implementation of indexing, we chose to index tracks.duration_ms and tracks.loudness. This turned out to be the most effective for filtering/sorting through the multiple columns. This is most likely because this method reduced the most rows that the query needed to examine before returning the results.

Command 4:

Initial EXPLAIN ANALYZE w/ No Indexing:

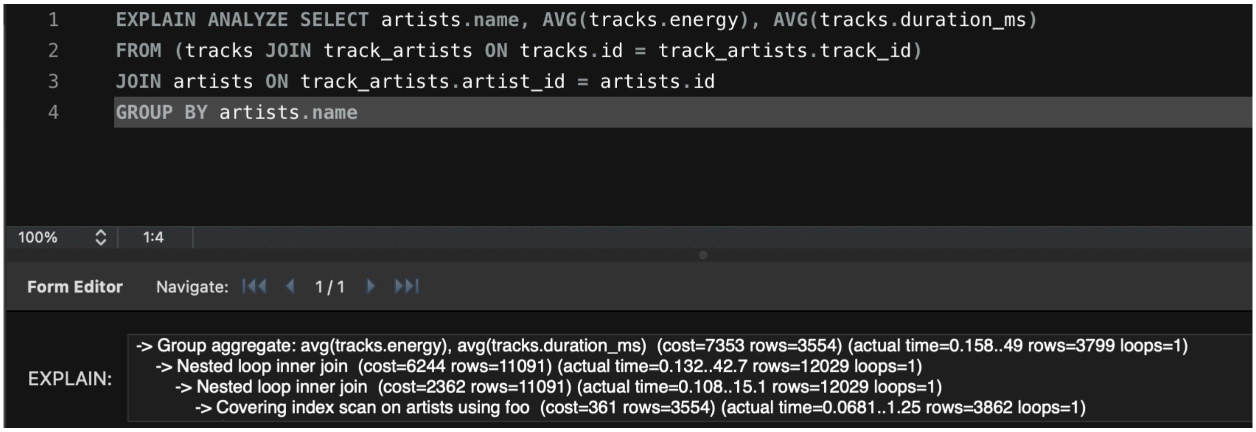


```
EXPLAIN ANALYZE SELECT artists.name, AVG(tracks.energy), AVG(tracks.duration_ms)
FROM (tracks JOIN track_artists ON tracks.id = track_artists.track_id)
JOIN artists ON track_artists.artist_id = artists.id
GROUP BY artists.name
```

```
-> Table scan on <temporary>  (actual time=58.5..59.4 rows=3799 loops=1)
   -> Aggregate using temporary table  (actual time=58.5..58.5 rows=3799 loops=1)
      -> Nested loop inner join  (cost=6244 rows=11091) (actual time=0.152..42.9 rows=12029 loops=1)
         -> Nested loop inner join  (cost=2362 rows=11091) (actual time=0.14..13.7 rows=12029 loops=1)
```

- **Cost: 6244 without indexing**

EXPLAIN ANALYZE with index on artists(name)

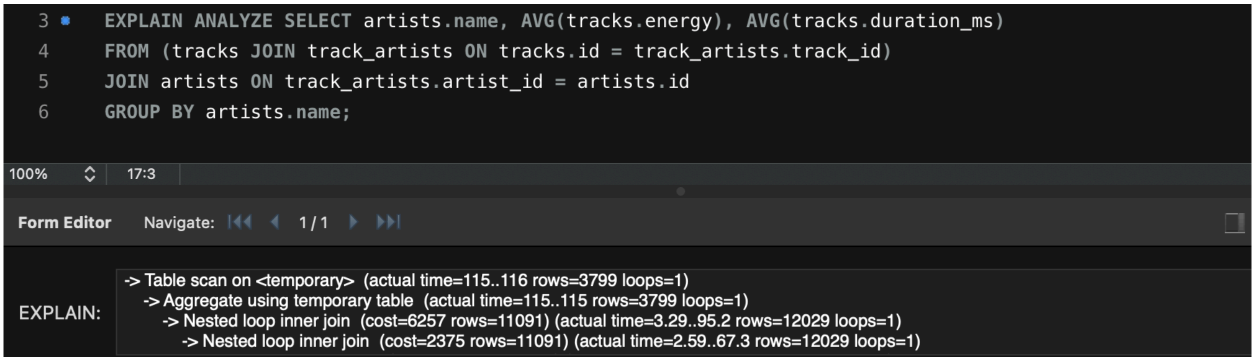


```
EXPLAIN ANALYZE SELECT artists.name, AVG(tracks.energy), AVG(tracks.duration_ms)
FROM (tracks JOIN track_artists ON tracks.id = track_artists.track_id)
JOIN artists ON track_artists.artist_id = artists.id
GROUP BY artists.name
```

```
-> Group aggregate: avg(tracks.energy), avg(tracks.duration_ms)  (cost=7353 rows=3554) (actual time=0.158..49 rows=3799 loops=1)
    -> Nested loop inner join  (cost=6244 rows=11091) (actual time=0.132..42.7 rows=12029 loops=1)
        -> Nested loop inner join  (cost=2362 rows=11091) (actual time=0.108..15.1 rows=12029 loops=1)
            -> Covering index scan on artists using foo  (cost=361 rows=3554) (actual time=0.0681..1.25 rows=3862 loops=1)
```

- **Cost: 6244**

<u>EXPLAIN ANALYZE with index on tracks(energy)</u>

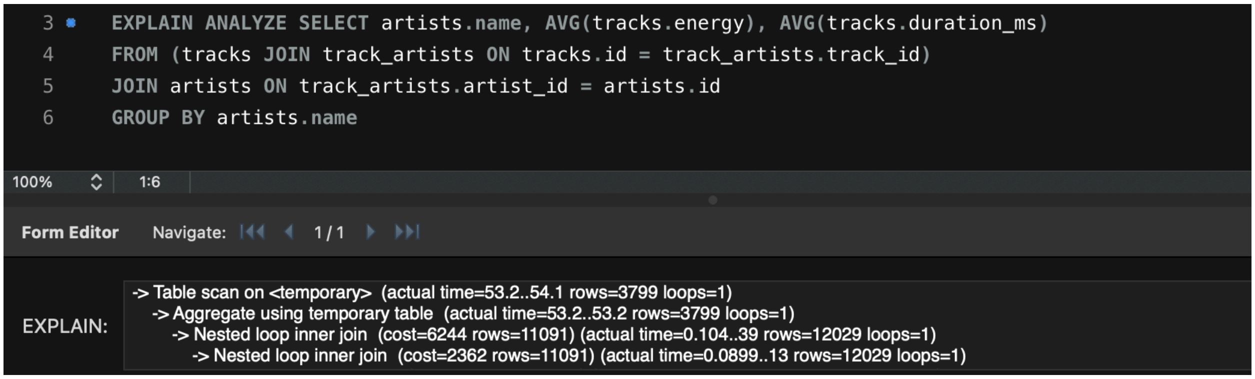


```
EXPLAIN ANALYZE SELECT artists.name, AVG(tracks.energy), AVG(tracks.duration_ms)
FROM (tracks JOIN track_artists ON tracks.id = track_artists.track_id)
JOIN artists ON track_artists.artist_id = artists.id
GROUP BY artists.name;
```

```
-> Table scan on <temporary>  (actual time=115..116 rows=3799 loops=1)
    -> Aggregate using temporary table  (actual time=115..115 rows=3799 loops=1)
        -> Nested loop inner join  (cost=6257 rows=11091) (actual time=3.29..95.2 rows=12029 loops=1)
            -> Nested loop inner join  (cost=2375 rows=11091) (actual time=2.59..67.3 rows=12029 loops=1)
```

- **Cost: 6257**

<u>EXPLAIN ANALYZE with index on tracks(duration_ms)</u>



```
EXPLAIN ANALYZE SELECT artists.name, AVG(tracks.energy), AVG(tracks.duration_ms)
FROM (tracks JOIN track_artists ON tracks.id = track_artists.track_id)
JOIN artists ON track_artists.artist_id = artists.id
GROUP BY artists.name
```

```
-> Table scan on <temporary>  (actual time=53.2..54.1 rows=3799 loops=1)
    -> Aggregate using temporary table  (actual time=53.2..53.2 rows=3799 loops=1)
        -> Nested loop inner join  (cost=6244 rows=11091) (actual time=0.104..39 rows=12029 loops=1)
            -> Nested loop inner join  (cost=2362 rows=11091) (actual time=0.0899..13 rows=12029 loops=1)
```

- **Cost: 6244**

<u>Indexing Analysis:</u> We chose to use indexing by the artist name as our final design. We tried indexing each attribute (and combinations of said attributes), and there was no improvement in

cost. We believe this is because the query itself is complex enough, so using an index does not improve the performance much, if at all. In addition, the index might not narrow down the result set significantly. In addition, the index may require a lot of maintenance.