Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

BIG DATA PROJECT PHASE - II
GROUP 34 - (SPOTIFY 12-M SONGS DATASET)

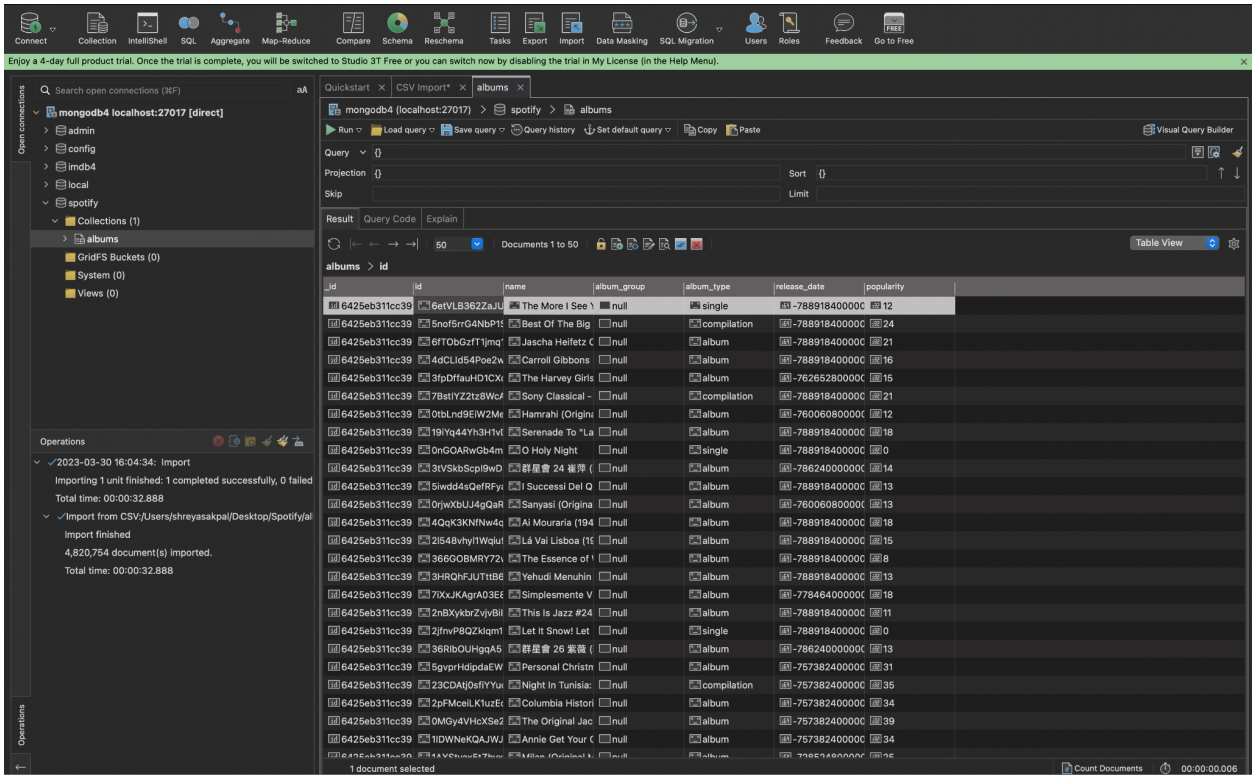
DOCUMENT - ORIENTED MODEL

1. Collection albums

```
db.createCollection("albums", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: [ "album_id", "name", "type", "release_date", "popularity" ],
      properties: {
        album_id: {
          bsonType: "string"
        },
        name: {
          bsonType: "string"
        },
        type: {
          bsonType: "string"
        },
        release_date: {
          bsonType: "int"
        },
        popularity: {
          bsonType: "int"
        }
      }
    }
  }
})
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)



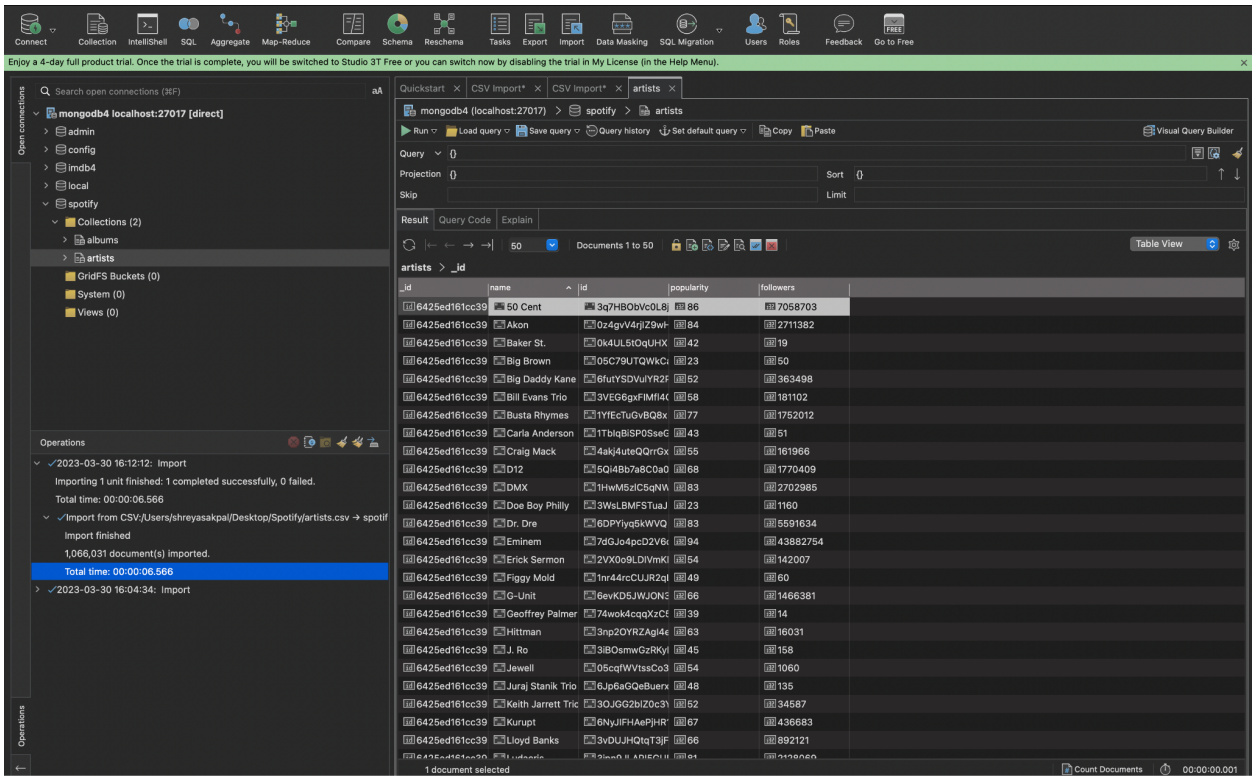
2. Collection artists

```
db.createCollection("artists", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         required: [ "artist_id", "name", "popularity", "followers" ],
         properties: {
            artist_id: {
               bsonType: "string"
            },
            name: {
               bsonType: "string"
            },
            popularity: {
               bsonType: "int"
            },
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

```
        followers: {
          bsonType: "int"
        }
      }
    }
  }
})
```



3. Collection audio_features

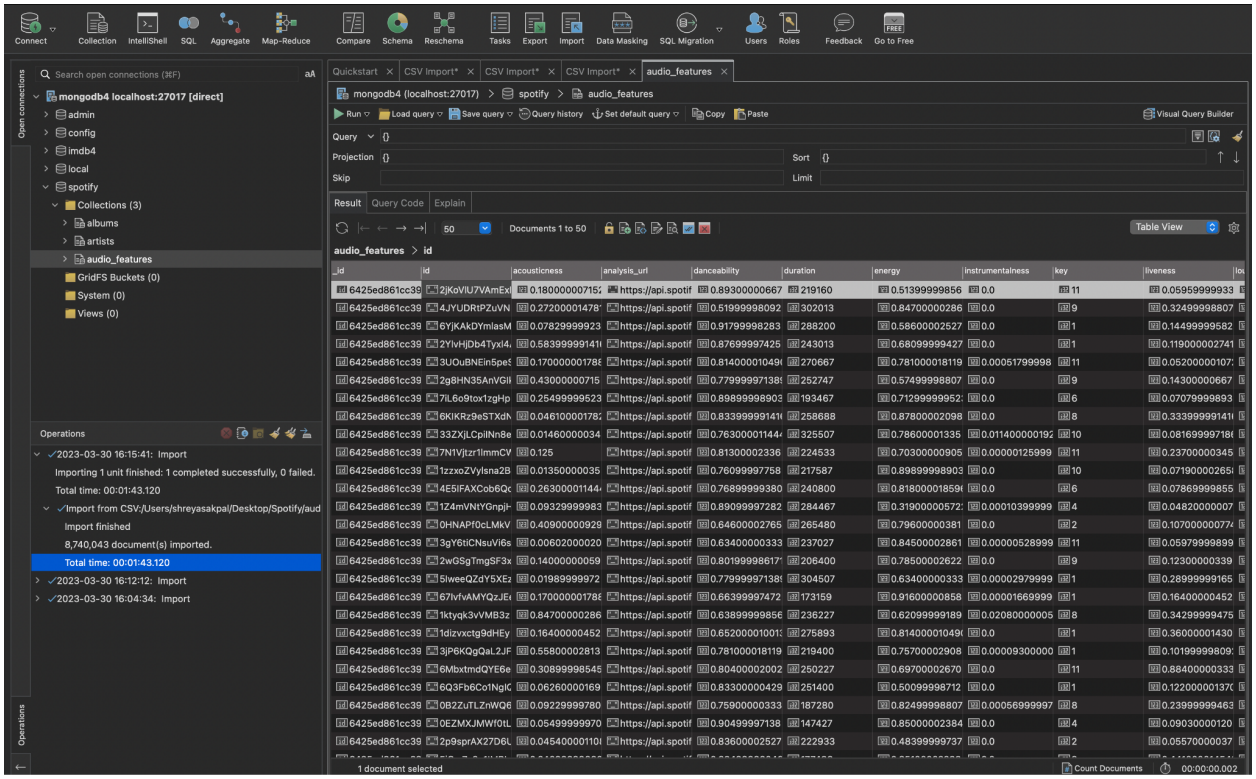
```
db.createCollection("audio_features", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: [ "audio_feature_id", "acousticness", "danceability", "duration", "energy",
"instrumentalness", "liveness", "loudness" ],
      properties: {
```

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

```
        audio_feature_id: {
          bsonType: "string"
        },
        acousticness: {
          bsonType: "decimal"
        },
        danceability: {
          bsonType: "decimal"
        },
        duration: {
          bsonType: "int"
        },
        energy: {
          bsonType: "decimal"
        },
        instrumentalness: {
          bsonType: "decimal"
        },
        liveness: {
          bsonType: "decimal"
        },
        loudness: {
          bsonType: "decimal"
        }
      }
    }
  }
})
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

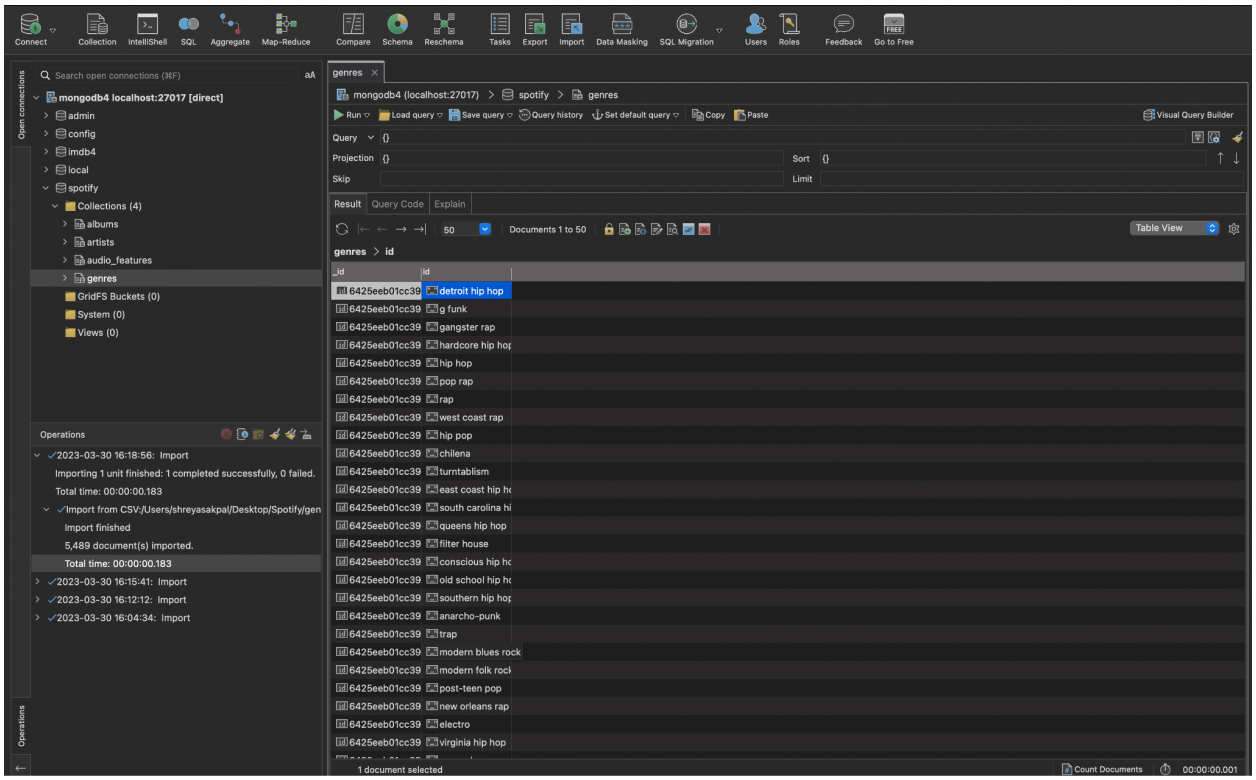


4. Collection genres

```
db.createCollection("genres", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         required: [ "genre_id", "name" ],
         properties: {
            genre_id: {
               bsonType: "int"
            },
            name: {
               bsonType: "string"
            }
         }
      }
   }
})
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

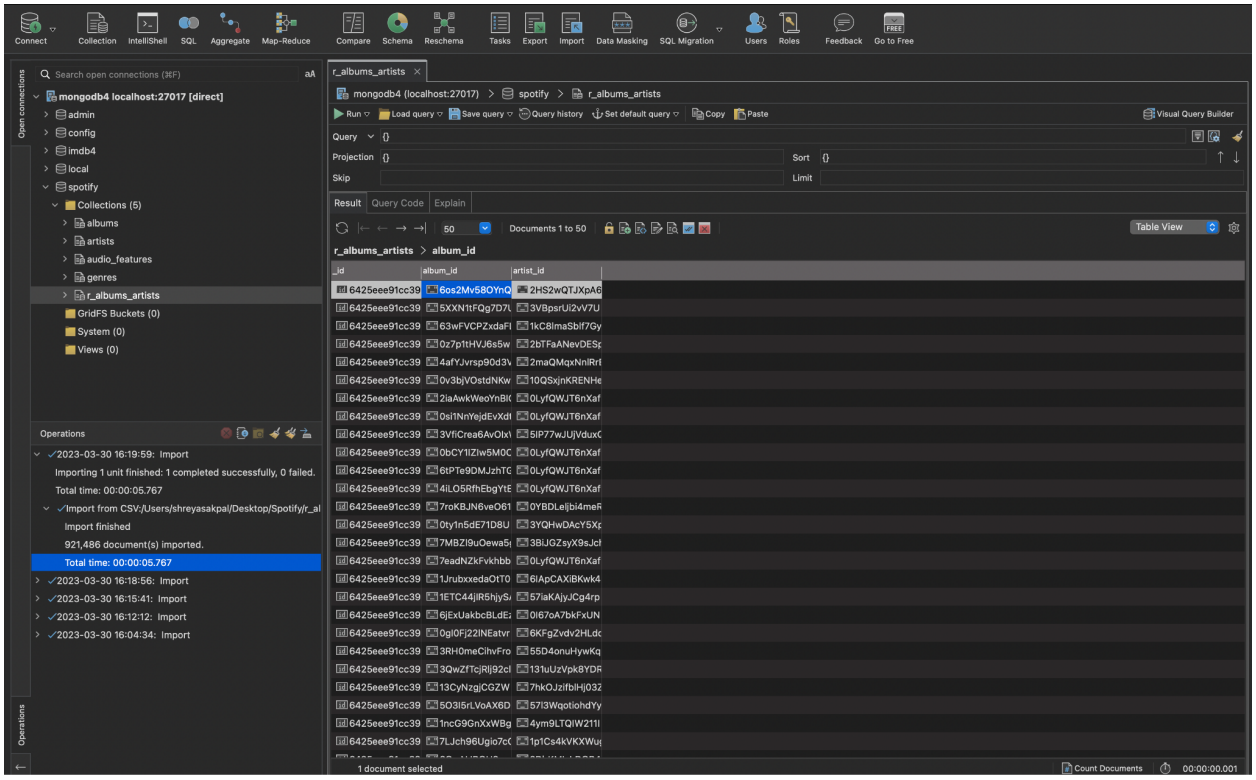


5. Collection r_albums_artists

```
db.createCollection("r_albums_artists", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         required: [ "album_id", "artist_id" ],
         properties: {
            album_id: {
               bsonType: "string"
            },
            artist_id: {
               bsonType: "string"
            }
         }
      }
```

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

```
  }
})
```

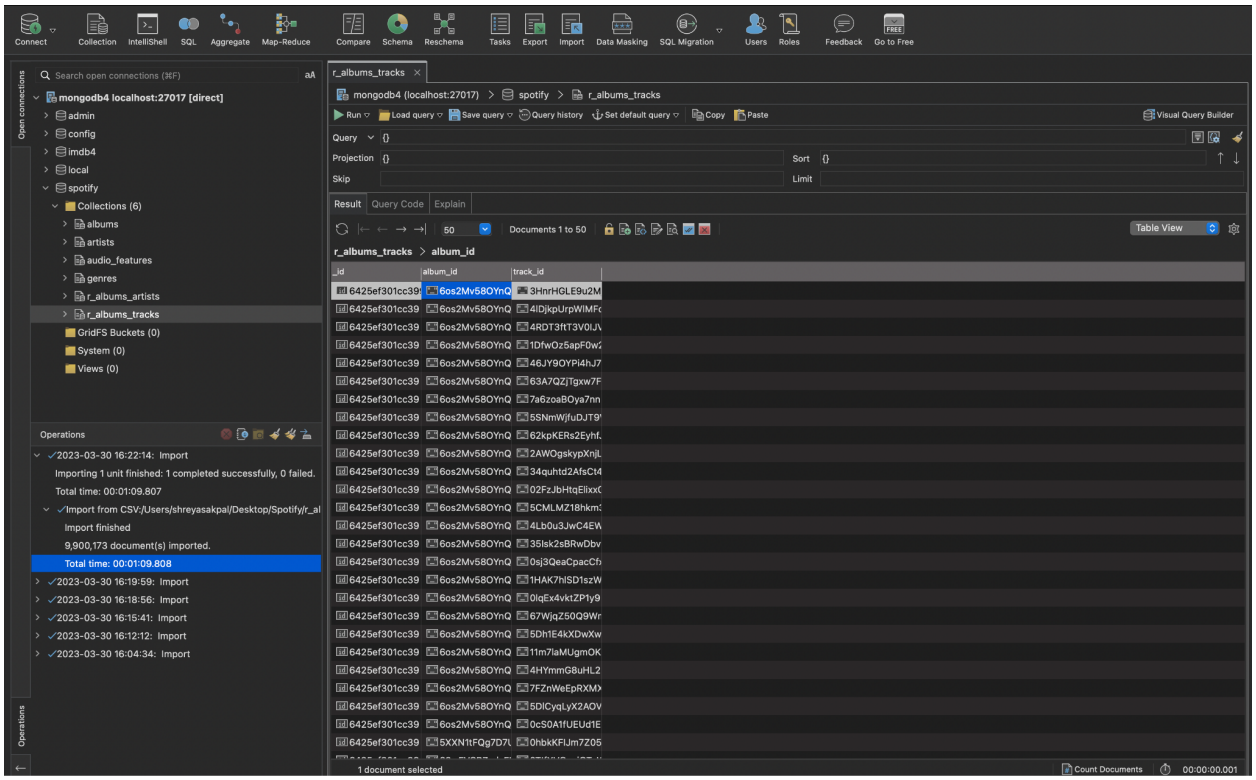


6. Collection r_albums_tracks

```
db.createCollection("r_albums_tracks", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: [ "album_id", "track_id" ],
      properties: {
        album_id: {
          bsonType: "string"
        },
        track_id: {
          bsonType: "string"
        }
```

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

```
        }
      }
    }
})
```

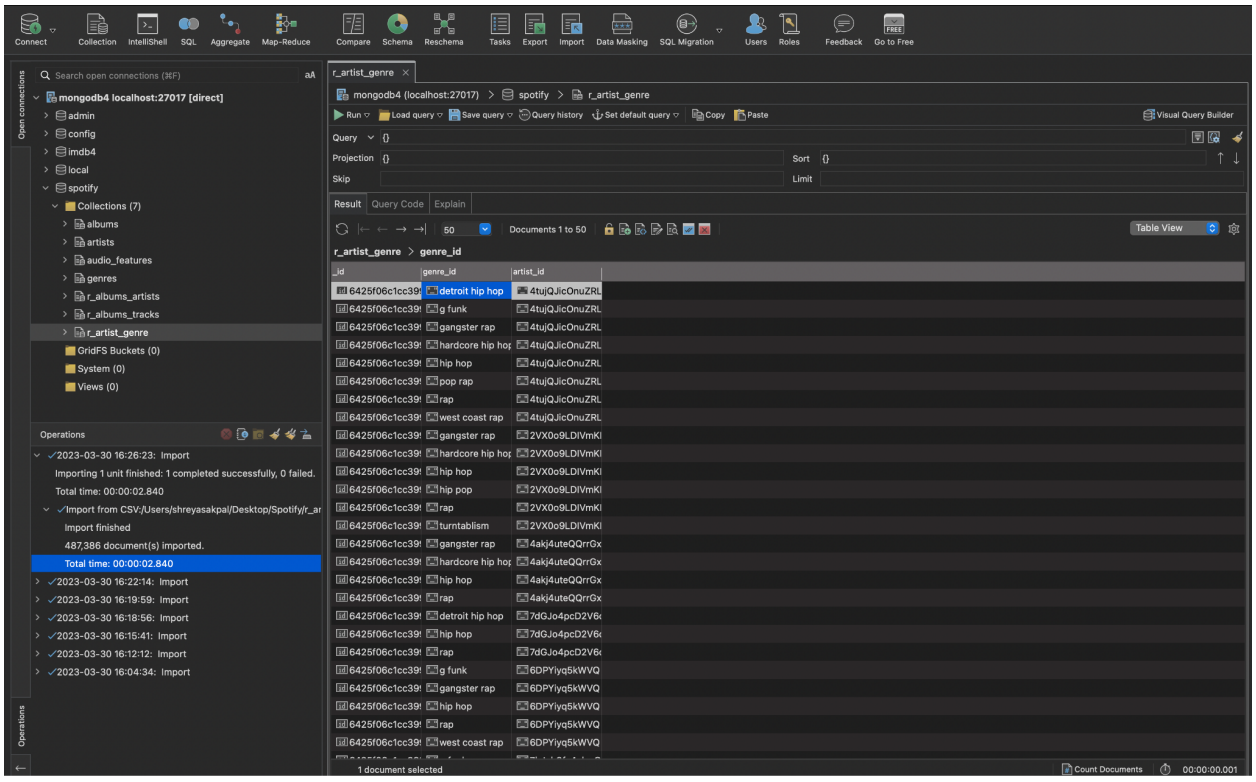


7. Collection r_artist_genre

```
db.createCollection("r_artist_genre", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: [ "artist_id", "genre_id" ],
      properties: {
        artist_id: {
          bsonType: "string",
          description: "must be a string and is required"
        },
```

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

```
        genre_id: {
          bsonType: "int",
          description: "must be an integer and is required"
        }
      }
    }
  }
})
```



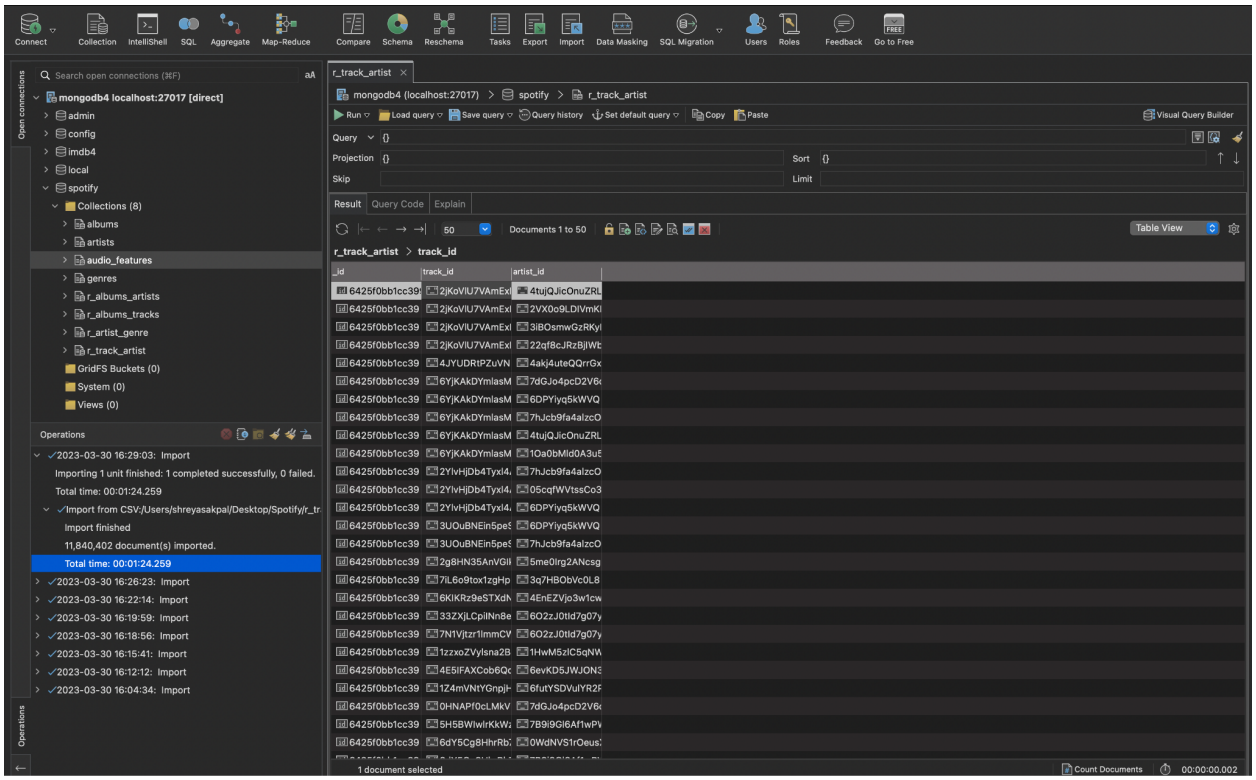
8. Collection r_track_artist

```
db.createCollection("r_track_artist", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: [ "track_id", "artist_id" ],
      properties: {
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

```
        track_id: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        artist_id: {
          bsonType: "string",
          description: "must be a string and is required"
        }
      }
    }
  }
})
```



9. Collection tracks

```
db.createCollection("tracks", {
  validator: {
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

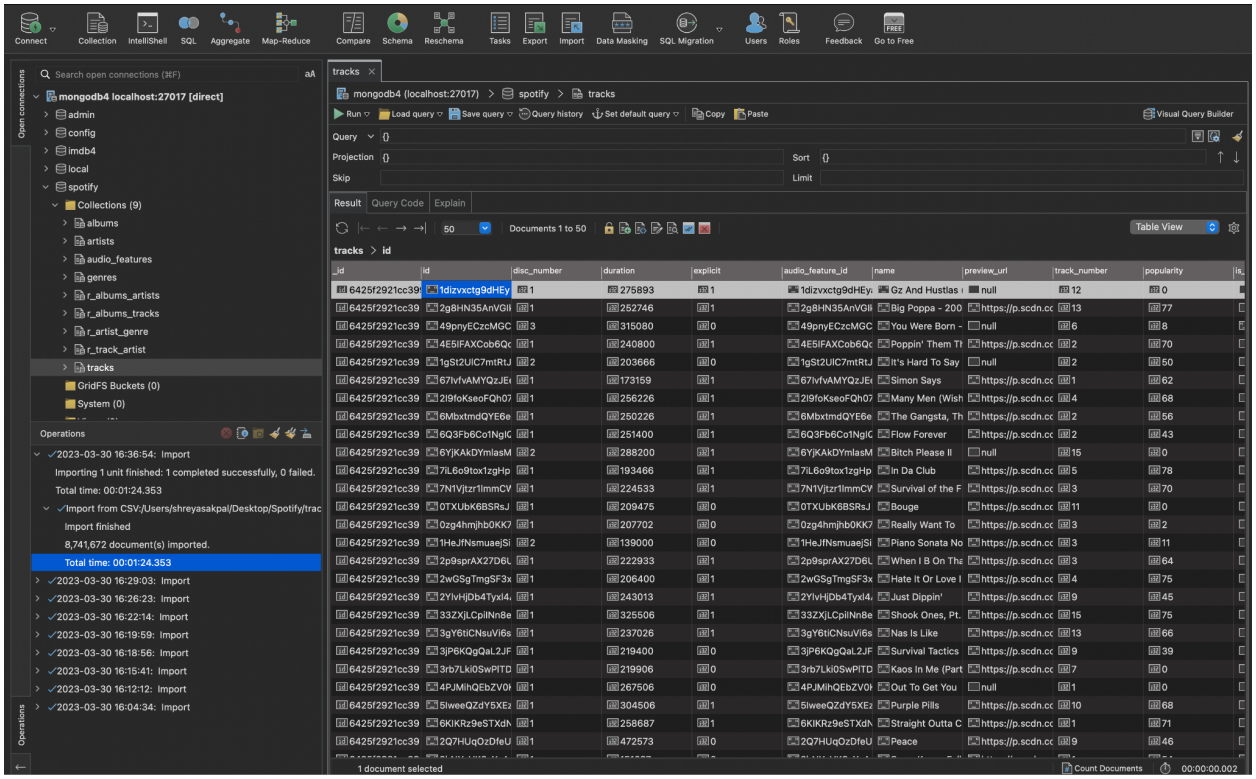
Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

```
    $jsonSchema: {
      bsonType: "object",
      required: [ "track_id", "name", "audio_feature_id", "popularity", "duration",
"explicit", "isPlayable", "track_no" ],
      properties: {
        track_id: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        name: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        audio_feature_id: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        popularity: {
          bsonType: "int",
          description: "must be an integer and is required"
        },
        duration: {
          bsonType: "int",
          description: "must be an integer and is required"
        },
        explicit: {
          bsonType: "bool",
          description: "must be a boolean and is required"
        },
        isPlayable: {
          bsonType: "bool",
          description: "must be a boolean and is required"
        },
        track_no: {
          bsonType: "int",
          description: "must be an integer and is required"
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

```
        }
      }
    }
  }
})
```



After creating the collections in MongoDB, we used the "Import Collection" option in Studio 3T to load data into the collections. This option allowed us to select the data source (a file or another collection), specify the format of the data, and any relevant options for the import process. Once the options were set, we initiated the import process, and Studio 3T loaded the data into the specified collections in the MongoDB database. The timing and loaded data is provided in the screenshots above.

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

COMPARING THE RELATIONAL MODEL AND THE DOCUMENT-BASED MODEL

The relational model organizes data into tables, with each table containing a primary key. This key uniquely identifies each row. The relationships between tables are established through foreign keys. This data is structured and its schema is defined before the data is added.

For example, a table was created “Tracks”, where the primary key was the track_id and a foreign key was added to establish a relationship between the table “Audio_Features” based on the audio_feature_id.

On the other hand, the document-based model stores the data in documents that are organized into collections. Each document is a self-contained unit that contains all the data and metadata associated with an object. This data can be structured or unstructured, and the schema is typically more flexible and dynamic than that of a relational database.

For example, to create a collection “Audio_Features”, we need to know the datatype of each field, whether it is decimal or integer or string, etc. In addition to that, we must also handle null values in a document-based model. To validate this condition, we mention all the fields which are required or cannot be null, while creating the collection.

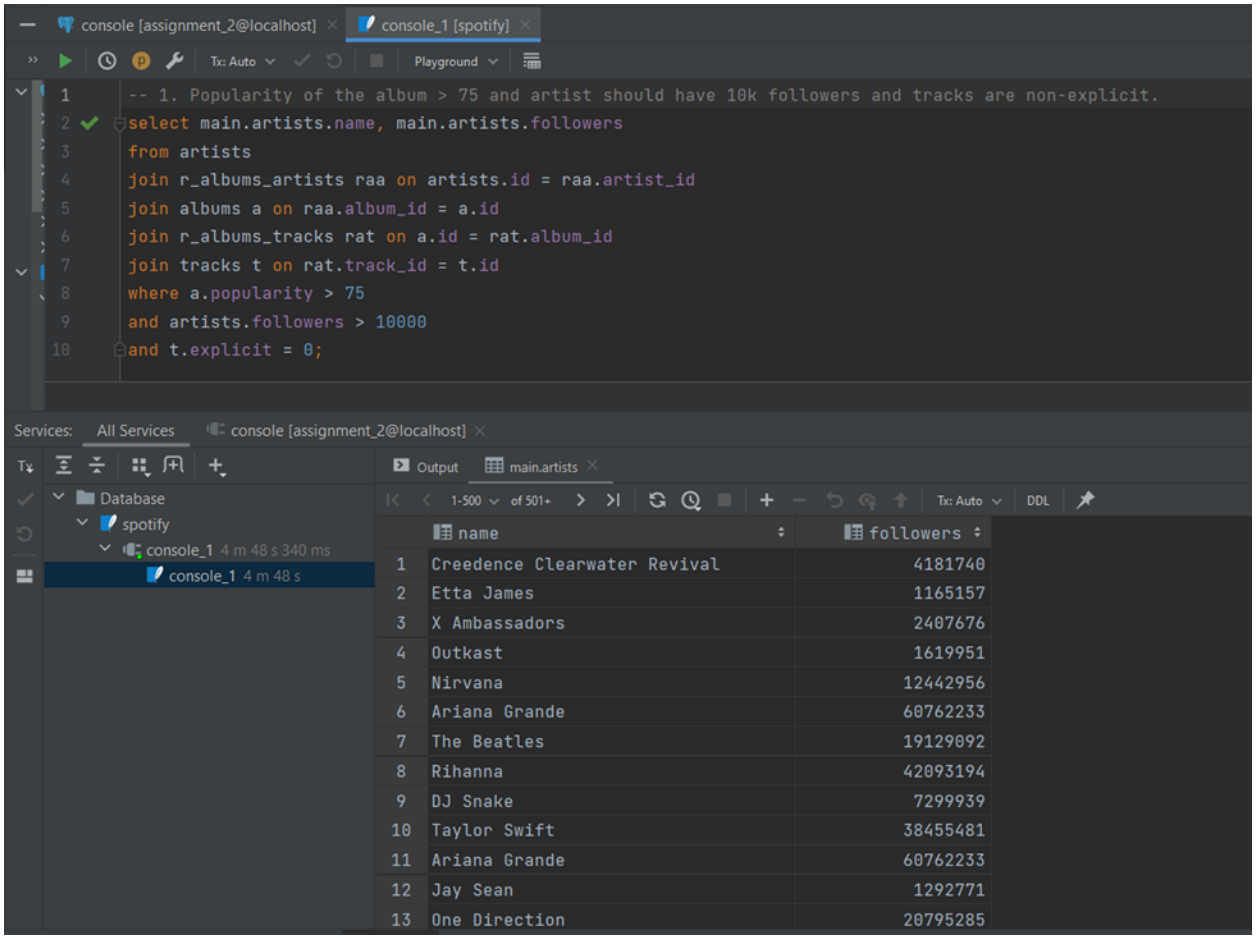
SQL QUERIES

1. Popularity of the album > 75 and artist should have 10k followers and tracks are non-explicit.

```
select main.artists.name, main.artists.followers
from artists
join r_albums_artists raa on artists.id = raa.artist_id
join albums a on raa.album_id = a.id
join r_albums_tracks rat on a.id = rat.album_id
join tracks t on rat.track_id = t.id
where a.popularity > 75
and artists.followers > 10000
and t.explicit = 0;
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)



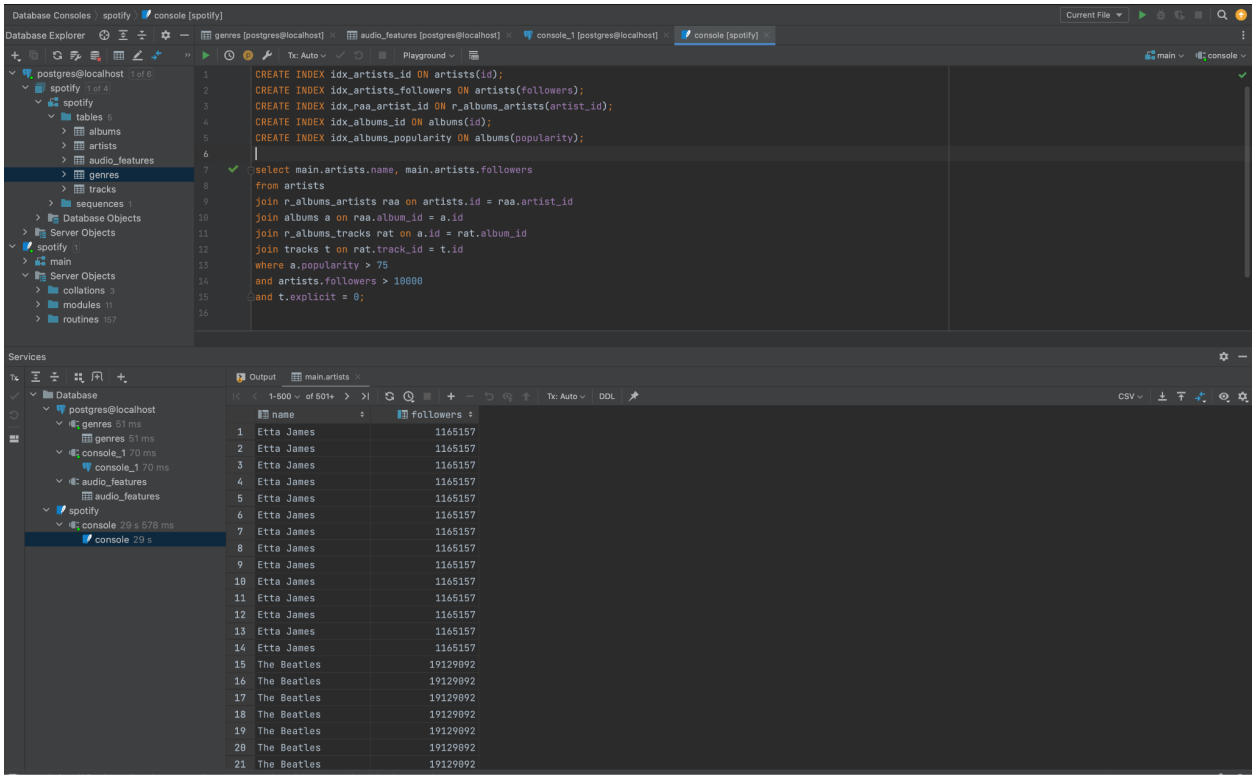
Time taken before Indexing: 4m 48s 340ms

Using Indexing:

```
CREATE INDEX idx_artists_id ON artists(id);
CREATE INDEX idx_artists_followers ON artists(followers);
CREATE INDEX idx_raa_artist_id ON r_albums_artists(artist_id);
CREATE INDEX idx_albums_id ON albums(id);
CREATE INDEX idx_albums_popularity ON albums(popularity);
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)



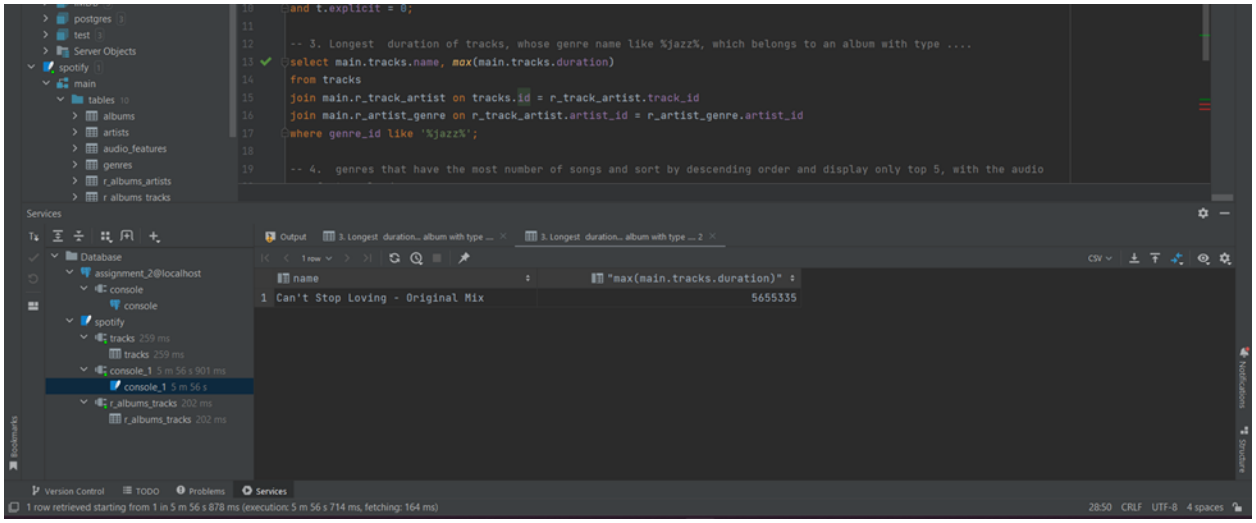
Time taken after Indexing: 29s 578ms

2. Longest duration of tracks, whose genre name like %jazz%

```
select main.tracks.name, max(main.tracks.duration)
from tracks
join main.r_track_artist on tracks.id = r_track_artist.track_id
join main.r_artist_genre on r_track_artist.artist_id = r_artist_genre.artist_id
where genre_id like '%jazz%';
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)



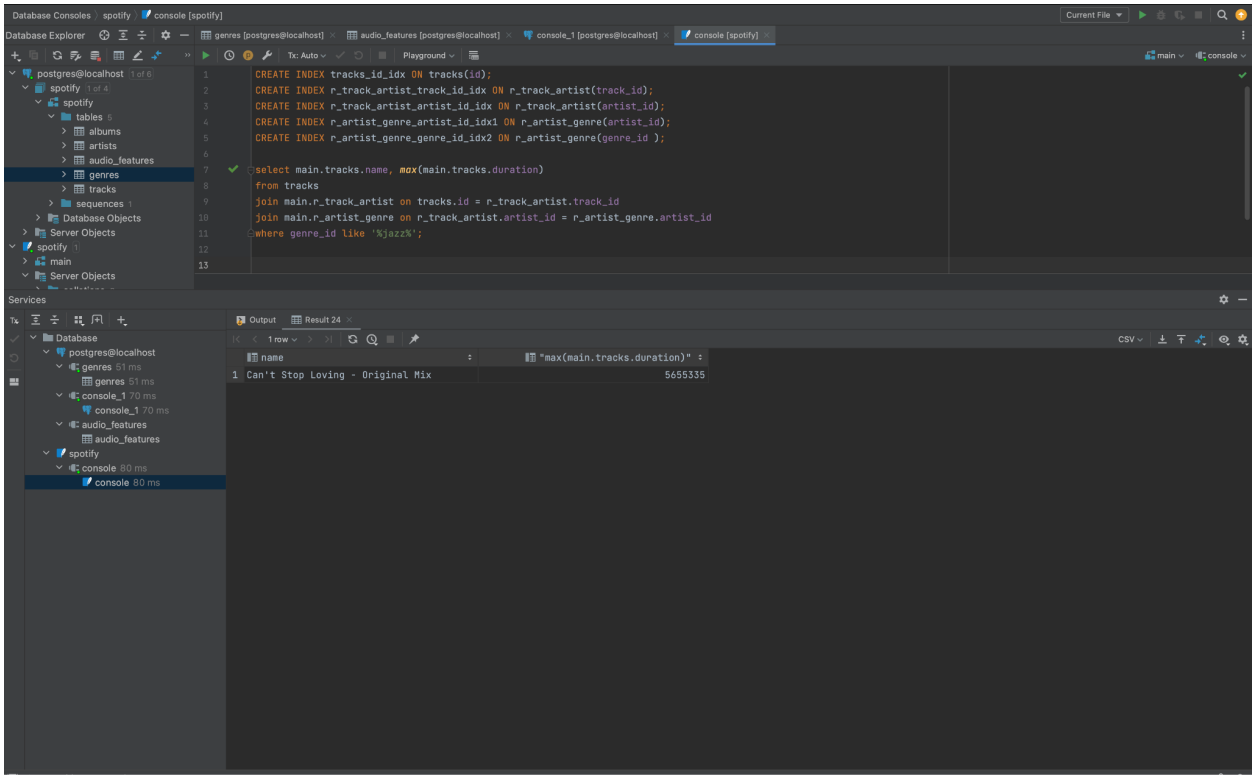
Time taken before Indexing: 5m 56s 901ms

Using Indexing:

```
CREATE INDEX tracks_id_idx ON tracks(id);
CREATE INDEX r_track_artist_track_id_idx ON r_track_artist(track_id);
CREATE INDEX r_track_artist_artist_id_idx ON r_track_artist(artist_id);
CREATE INDEX r_artist_genre_artist_id_idx1 ON r_artist_genre(artist_id);
CREATE INDEX r_artist_genre_genre_id_idx2 ON r_artist_genre(genre_id );
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)



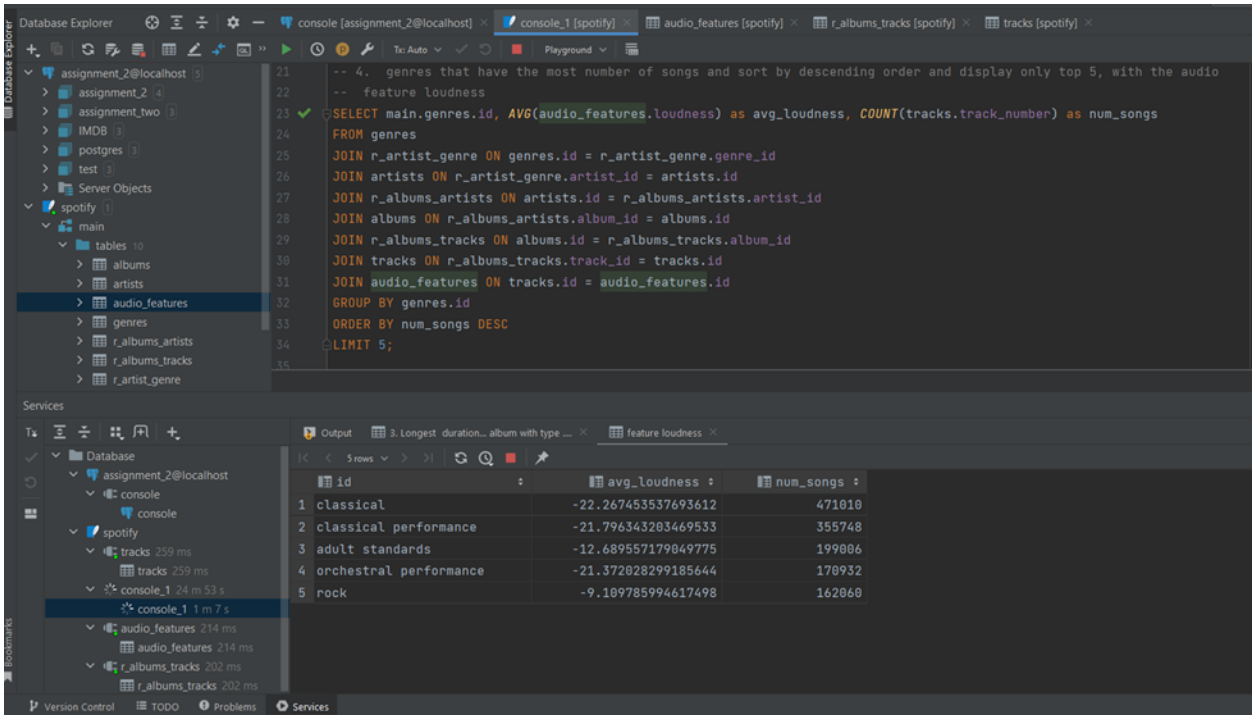
Time taken after Indexing: 80ms

3. Genres that have the most number of songs and sort by descending order and display only top 5, with the audio feature loudness

```
SELECT main.genres.id, AVG(audio_features.loudness) as avg_loudness,
COUNT(tracks.track_number) as num_songs
FROM genres
JOIN r_artist_genre ON genres.id = r_artist_genre.genre_id
JOIN artists ON r_artist_genre.artist_id = artists.id
JOIN r_albums_artists ON artists.id = r_albums_artists.artist_id
JOIN albums ON r_albums_artists.album_id = albums.id
JOIN r_albums_tracks ON albums.id = r_albums_tracks.album_id
JOIN tracks ON r_albums_tracks.track_id = tracks.id
JOIN audio_features ON tracks.id = audio_features.id
GROUP BY genres.id
ORDER BY num_songs DESC
LIMIT 5;
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)



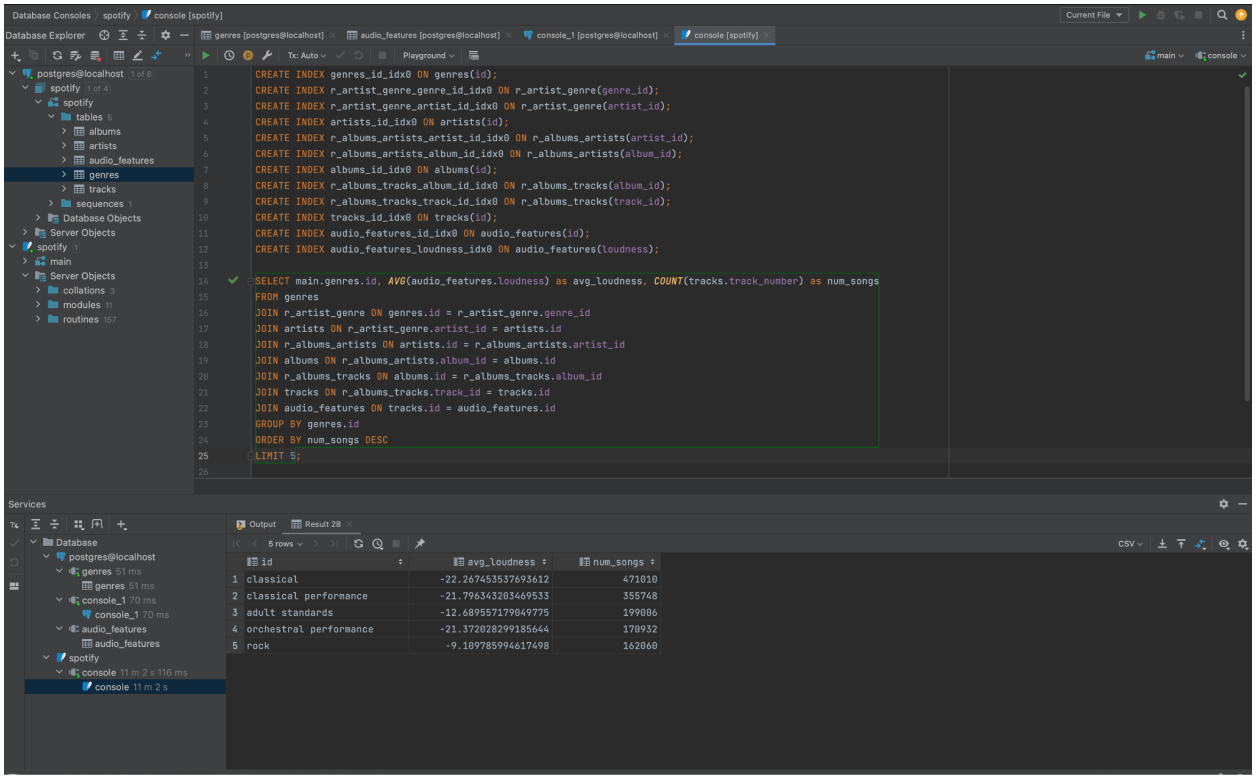
Time taken before Indexing: 24m 53ms

Using Indexing:

```
CREATE INDEX genres_id_idx0 ON genres(id);
CREATE INDEX r_artist_genre_genre_id_idx0 ON r_artist_genre(genre_id);
CREATE INDEX r_artist_genre_artist_id_idx0 ON r_artist_genre(artist_id);
CREATE INDEX artists_id_idx0 ON artists(id);
CREATE INDEX r_albums_artists_artist_id_idx0 ON r_albums_artists(artist_id);
CREATE INDEX r_albums_artists_album_id_idx0 ON r_albums_artists(album_id);
CREATE INDEX albums_id_idx0 ON albums(id);
CREATE INDEX r_albums_tracks_album_id_idx0 ON r_albums_tracks(album_id);
CREATE INDEX r_albums_tracks_track_id_idx0 ON r_albums_tracks(track_id);
CREATE INDEX tracks_id_idx0 ON tracks(id);
CREATE INDEX audio_features_id_idx0 ON audio_features(id);
CREATE INDEX audio_features_loudness_idx0 ON audio_features(loudness);
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)



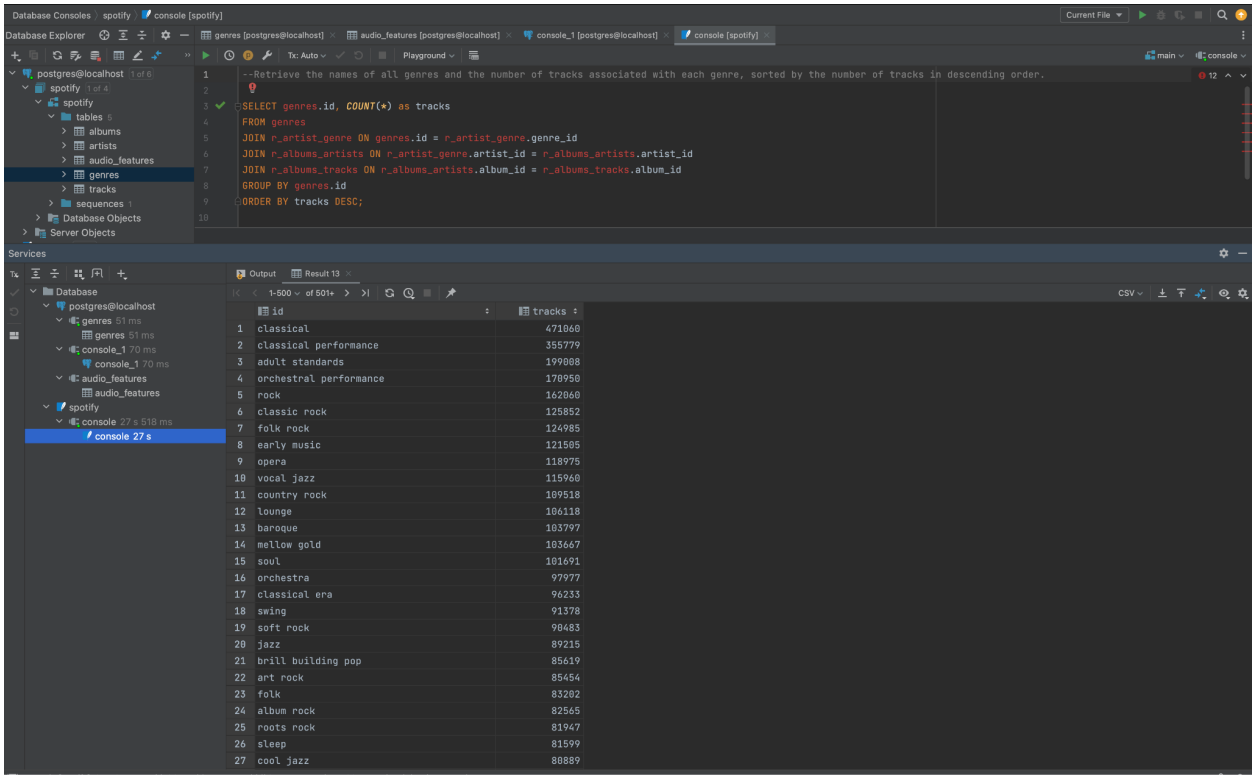
Time taken after Indexing: 11m 2s 116ms

4. Retrieve the names of all genres and the number of tracks associated with each genre, sorted by the number of tracks in descending order.

```
SELECT genres.id, COUNT(*) as tracks
FROM genres
JOIN r_artist_genre ON genres.id = r_artist_genre.genre_id
JOIN r_albums_artists ON r_artist_genre.artist_id = r_albums_artists.artist_id
JOIN r_albums_tracks ON r_albums_artists.album_id = r_albums_tracks.album_id
GROUP BY genres.id
ORDER BY tracks DESC;
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)



	id	tracks
1	classical	471060
2	classical performance	355779
3	adult standards	199008
4	orchestral performance	170950
5	rock	162060
6	classic rock	125852
7	folk rock	124985
8	early music	121505
9	opera	118975
10	vocal jazz	115960
11	country rock	109518
12	lounge	106118
13	baroque	103797
14	mellow gold	103667
15	soul	101691
16	orchestra	97977
17	classical era	96233
18	swing	91378
19	soft rock	90483
20	jazz	89215
21	brill building pop	85619
22	art rock	85454
23	folk	83202
24	album rock	82565
25	roots rock	81947
26	sleep	81599
27	cool jazz	80889

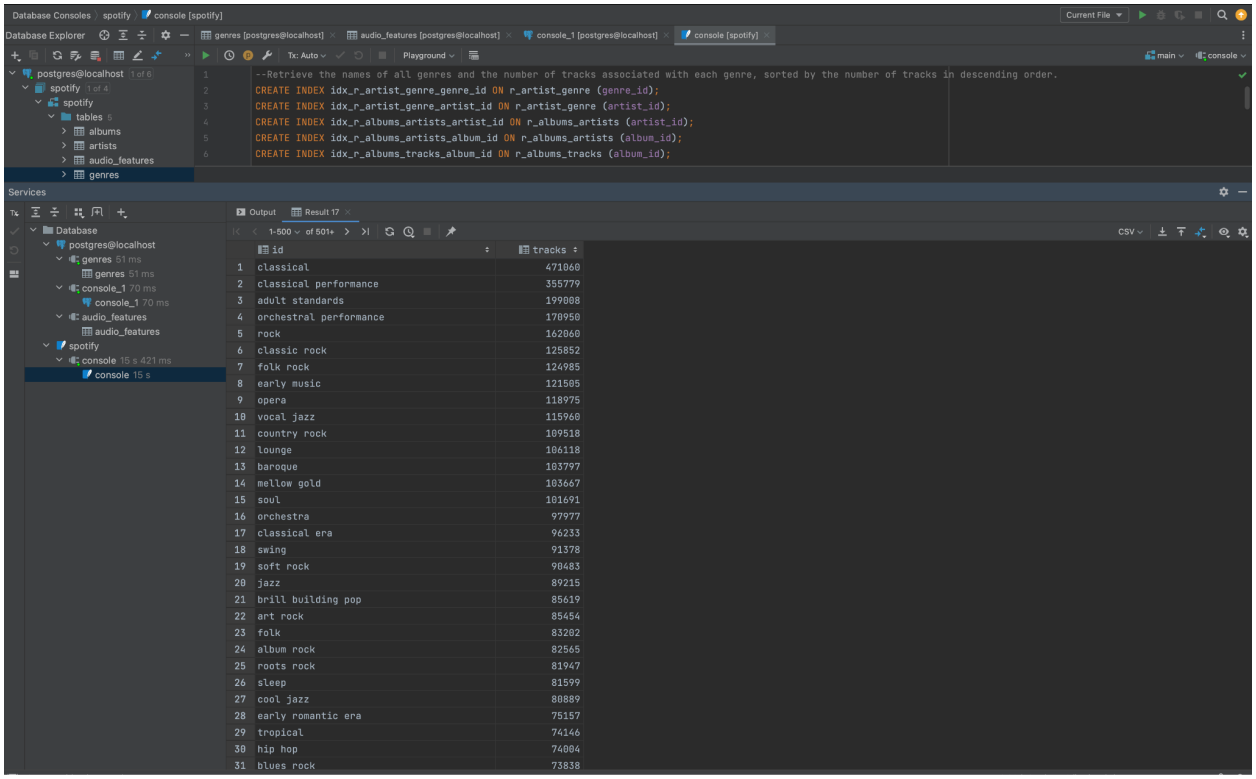
Time taken before Indexing: 27s 518ms

Using Indexing:

```
CREATE INDEX idx_r_artist_genre_genre_id ON r_artist_genre (genre_id);
CREATE INDEX idx_r_artist_genre_artist_id ON r_artist_genre (artist_id);
CREATE INDEX idx_r_albums_artists_artist_id ON r_albums_artists (artist_id);
CREATE INDEX idx_r_albums_artists_album_id ON r_albums_artists (album_id);
CREATE INDEX idx_r_albums_tracks_album_id ON r_albums_tracks (album_id);
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)



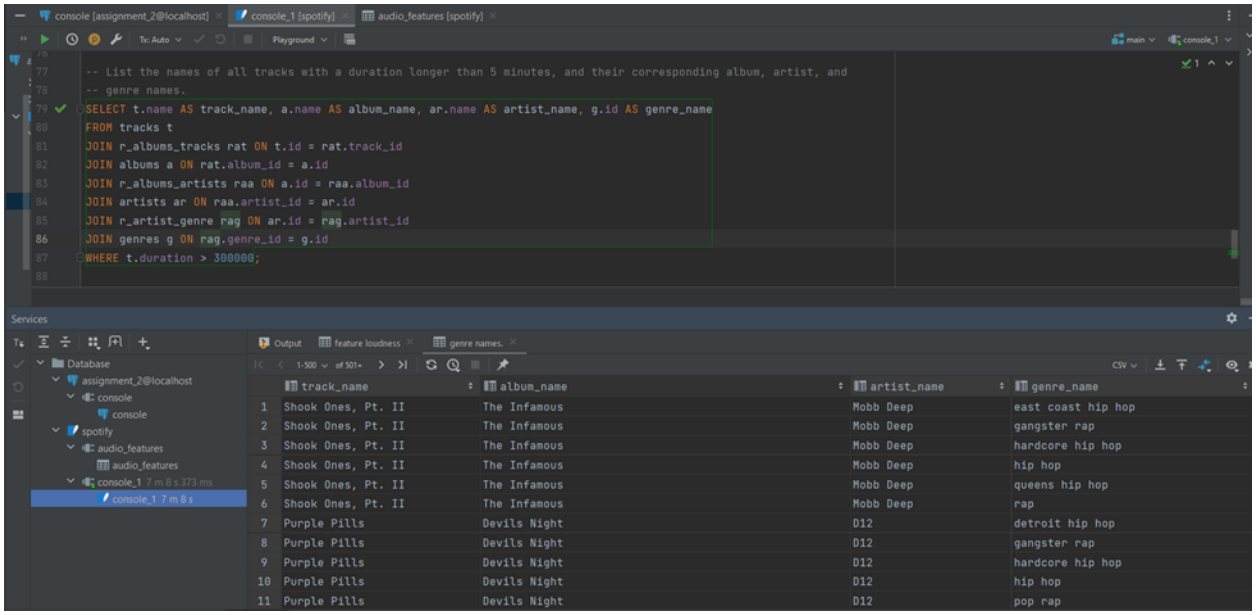
Time taken after Indexing: 15s 421ms

5. List the names of all tracks with a duration longer than 5 minutes, and their corresponding album, artist, and genre names.

```
SELECT t.name AS track_name, a.name AS album_name, ar.name AS artist_name, g.id
AS genre_name
FROM tracks t
JOIN r_albums_tracks rat ON t.id = rat.track_id
JOIN albums a ON rat.album_id = a.id
JOIN r_albums_artists raa ON a.id = raa.album_id
JOIN artists ar ON raa.artist_id = ar.id
JOIN r_artist_genre rag ON ar.id = rag.artist_id
JOIN genres g ON rag.genre_id = g.id
WHERE t.duration > 300000;
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)



```
-- List the names of all tracks with a duration longer than 5 minutes, and their corresponding album, artist, and
-- genre names.
SELECT t.name AS track_name, a.name AS album_name, ar.name AS artist_name, g.id AS genre_name
FROM tracks t
JOIN r_albums_tracks rat ON t.id = rat.track_id
JOIN albums a ON rat.album_id = a.id
JOIN r_albums_artists raa ON a.id = raa.album_id
JOIN artists ar ON raa.artist_id = ar.id
JOIN r_artist_genre rag ON ar.id = rag.artist_id
JOIN genres g ON rag.genre_id = g.id
WHERE t.duration > 300000;
```

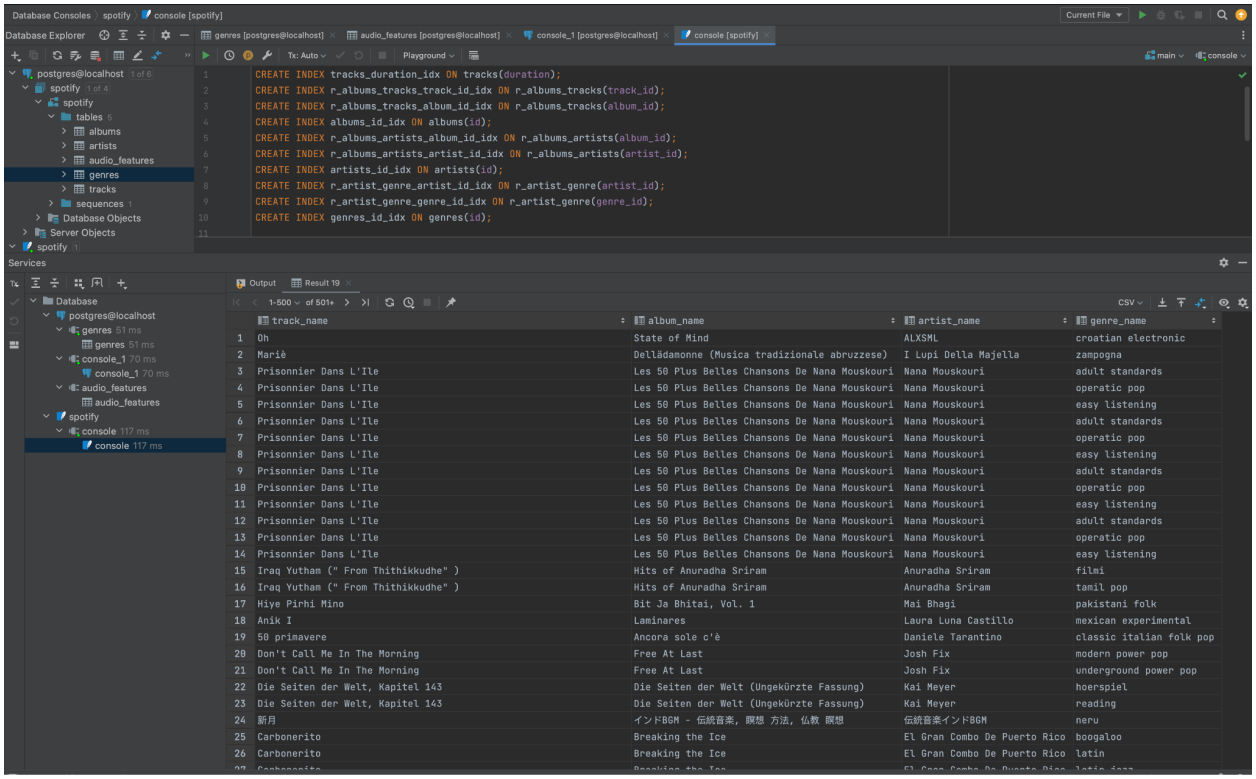
	track_name	album_name	artist_name	genre_name
1	Shook Ones, Pt. II	The Infamous	Mobb Deep	east coast hip hop
2	Shook Ones, Pt. II	The Infamous	Mobb Deep	gangster rap
3	Shook Ones, Pt. II	The Infamous	Mobb Deep	hardcore hip hop
4	Shook Ones, Pt. II	The Infamous	Mobb Deep	hip hop
5	Shook Ones, Pt. II	The Infamous	Mobb Deep	queens hip hop
6	Shook Ones, Pt. II	The Infamous	Mobb Deep	rap
7	Purple Pills	Devils Night	D12	detroit hip hop
8	Purple Pills	Devils Night	D12	gangster rap
9	Purple Pills	Devils Night	D12	hardcore hip hop
10	Purple Pills	Devils Night	D12	hip hop
11	Purple Pills	Devils Night	D12	pop rap

Using Indexing:

```
CREATE INDEX tracks_duration_idx ON tracks(duration);
CREATE INDEX r_albums_tracks_track_id_idx ON r_albums_tracks(track_id);
CREATE INDEX r_albums_tracks_album_id_idx ON r_albums_tracks(album_id);
CREATE INDEX albums_id_idx ON albums(id);
CREATE INDEX r_albums_artists_album_id_idx ON r_albums_artists(album_id);
CREATE INDEX r_albums_artists_artist_id_idx ON r_albums_artists(artist_id);
CREATE INDEX artists_id_idx ON artists(id);
CREATE INDEX r_artist_genre_artist_id_idx ON r_artist_genre(artist_id);
CREATE INDEX r_artist_genre_genre_id_idx ON r_artist_genre(genre_id);
CREATE INDEX genres_id_idx ON genres(id);
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)



	track_name	album_name	artist_name	genre_name
1	Oh	State of Mind	ALXSML	croatian electronic
2	Marìè	Dellädamonne (Musica tradizionale abruzzese)	I Lupi Della Majella	zampogna
3	Prisonnier Dans L'Ile	Les 50 Plus Belles Chansons De Nana Mouskouri	Nana Mouskouri	adult standards
4	Prisonnier Dans L'Ile	Les 50 Plus Belles Chansons De Nana Mouskouri	Nana Mouskouri	operatic pop
5	Prisonnier Dans L'Ile	Les 50 Plus Belles Chansons De Nana Mouskouri	Nana Mouskouri	easy listening
6	Prisonnier Dans L'Ile	Les 50 Plus Belles Chansons De Nana Mouskouri	Nana Mouskouri	adult standards
7	Prisonnier Dans L'Ile	Les 50 Plus Belles Chansons De Nana Mouskouri	Nana Mouskouri	operatic pop
8	Prisonnier Dans L'Ile	Les 50 Plus Belles Chansons De Nana Mouskouri	Nana Mouskouri	easy listening
9	Prisonnier Dans L'Ile	Les 50 Plus Belles Chansons De Nana Mouskouri	Nana Mouskouri	adult standards
10	Prisonnier Dans L'Ile	Les 50 Plus Belles Chansons De Nana Mouskouri	Nana Mouskouri	operatic pop
11	Prisonnier Dans L'Ile	Les 50 Plus Belles Chansons De Nana Mouskouri	Nana Mouskouri	easy listening
12	Prisonnier Dans L'Ile	Les 50 Plus Belles Chansons De Nana Mouskouri	Nana Mouskouri	adult standards
13	Prisonnier Dans L'Ile	Les 50 Plus Belles Chansons De Nana Mouskouri	Nana Mouskouri	operatic pop
14	Prisonnier Dans L'Ile	Les 50 Plus Belles Chansons De Nana Mouskouri	Nana Mouskouri	easy listening
15	Iraq Yutham (" From Thithikkudhe")	Hits of Anuradha Sriram	Anuradha Sriram	filmi
16	Iraq Yutham (" From Thithikkudhe")	Hits of Anuradha Sriram	Anuradha Sriram	tamil pop
17	Hiye Pirhi Mino	Bit Ja Bhitai, Vol. 1	Mai Bhagi	pakistani folk
18	Anik I	Laminares	Laura Luna Castillo	mexican experimental
19	50 primavere	Ancora sole c'è	Daniele Tarantino	classic italian folk pop
20	Don't Call Me In The Morning	Free At Last	Josh Fix	modern power pop
21	Don't Call Me In The Morning	Free At Last	Josh Fix	underground power pop
22	Die Seiten der Welt, Kapitel 143	Die Seiten der Welt (Ungekürzte Fassung)	Kai Meyer	hoerspiel
23	Die Seiten der Welt, Kapitel 143	Die Seiten der Welt (Ungekürzte Fassung)	Kai Meyer	reading
24	新月	インドBGM - 伝統音楽, 瞑想 方法, 仏教 瞑想	伝統音楽インドBGM	neru
25	Carbonerito	Breaking the Ice	El Gran Combo De Puerto Rico	boogaloo
26	Carbonerito	Breaking the Ice	El Gran Combo De Puerto Rico	latin

Time taken after Indexing: 117ms

FUNCTIONAL DEPENDENCIES AND NORMALIZATION

A functional dependency exists when one or more attributes in a table uniquely determines the value of another attribute. We have identified several functional dependencies (FDs) between attributes in the tables given as follows:

- **artist_id -> artist_name, popularity, followers**

  This means that the artist_id uniquely determines the artist_name, popularity, and followers attributes. In other words, for each artist_id, there can be only one artist_name, popularity, and followers.

- **artist_name, popularity, followers -> artist_id**

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

This means that given the artist_name, popularity, and followers attributes, we can uniquely determine the artist_id. In other words, each combination of artist_name, popularity, and followers corresponds to only one artist_id.

- **album_id -> album_name, album_type, release_date, popularity**

  This means that the album_id uniquely determines the album_name, album_type, release_date, and popularity attributes. In other words, for each album_id, there can be only one album_name, album_type, release_date, and popularity.

- **album_name, album_type, release_date, popularity -> album_id**

  This means that given the album_name, album_type, release_date, and popularity attributes, we can uniquely determine the album_id. In other words, each combination of album_name, album_type, release_date, and popularity corresponds to only one album_id.

- **album_name, release_date -> album_id, album_type, popularity**

  In the albums table, given the values of album_name and release_date, we can determine the values of album_id, album_type, and popularity. In other words, the combination of album_name and release_date uniquely identifies an album and its associated attributes.

- **track_id -> name, audio_feature_id, popularity, duration, explicit, isPlayable, track_no**

  This means that the track_id uniquely determines the name, audio_feature_id, popularity, duration, explicit, isPlayable, and track_no attributes. In other words, for each track_id, there can be only one name, audio_feature_id, popularity, duration, explicit, isPlayable, and track_no.

- **name, duration, audio_feature_id, track_no -> track_id, explicit, isPlayable**

  This means that given the name, duration, audio_feature_id, and track_no attributes, we can determine the track_id, explicit, and isPlayable. In other words, each combination of name, duration, audio_feature_id, and track_no corresponds to only one track_id, explicit, and isPlayable.

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

- **name, popularity, duration, track_no -> track_id, explicit, isPlayable**

  In the Tracks table, given the values of name, popularity of the track, duration and the track number that the track was given in the album, we can uniquely identify the track and its attributes such as if it is playable or explicit.

- **audio_feature_id, name -> track_id**

  This means that given the audio_feature_id and name attributes, we can uniquely determine the track_id. In other words, each combination of audio_feature_id and name corresponds to only one track_id.

- **genre_id -> genre_name**

  This means that the genre_id uniquely determines the genre_name. In other words, for each genre_id, there can be only one genre_name.

- **audio_feature_id -> acousticness, danceability, duration, energy, instrumentalness, liveness, loudness**

  Similarly, in the Audio features table, given the values of the audio_feature_id, we can uniquely identify all the above attributes such as loudness, energy, liveness, duration etc. for a given track.

- **acousticness, danceability, duration, energy, instrumentalness, liveness, loudness -> audio_feature_id**

  This means that given the acousticness, danceability, duration, energy, instrumentalness, liveness, and loudness attributes, we can uniquely determine the audio_feature_id. In other words, each combination of acousticness, danceability, duration, energy, instrumentalness, liveness, and loudness corresponds to only one audio_feature_id.

- **album_id -> artist_id**

  This means that the album_id uniquely determines the artist_id. In other words, for each album_id, there can be only one artist_id.

- **album_id -> track_id**

Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

This means that the album_id uniquely determines the track_id. In other words, for each album_id, there can be only one track_id.

- **artist_id -> genre_id**

  This means that the artist_id uniquely determines the genre_id. In other words, for each artist_id, there can be only one genre_id.

- **track_id -> artist_id**

  This means that the track_id uniquely determines the artist_id. In other words, for each track_id, there can be only one artist_id.

The objective of normalization is to eliminate data anomalies that can arise due to data redundancy and inconsistency. There are different levels of normalization, including 1st Normal Form (1NF), 2nd Normal Form (2NF), and 3rd Normal Form (3NF).

In our relational model, all tables are already in 1NF, which means that all attributes contain atomic values and there are no repeating groups. Furthermore, all tables are also in 2NF, which means that all non-key attributes are functionally dependent on the entire primary key.

To achieve 3NF, we have further normalized the Tracks table by separating the explicit and isPlayable attributes into a separate table called Track-Details.

Track-Details table:
- track_id
- explicit
- isPlayable

This ensures that the tracks table only contains attributes that are directly related to the track's audio features or its name, and hence, it is in 3NF and that there is no redundancy or inconsistency in the data.

Overall, by identifying functional dependencies and normalizing the tables in our relational model, we have ensured that the data is organized in a way that is efficient, consistent, and easy to maintain. This helps to improve data quality and reduces the risk of data anomalies arising in the database.