

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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)

The screenshot shows the Studio 3T interface with the following details:

- Connections:** Open connection to `mongodb4 (localhost:27017 [direct])`.
- Operations:**
 - Imported 4,820,754 documents from CSV.
 - Total time: 00:00:32.888.
- Results:** Shows the `albums` collection with 4,820,754 documents. The table includes columns: `_id`, `id`, `name`, `album_group`, `album_type`, `release_date`, and `popularity`. A sample of documents is shown, such as:
 - `6425eb31cc39`: `6etvLB362ZaJl`, `The More I See`, `null`, `single`, `-788918400000`, `12`
 - `6425eb31cc39`: `5nof5rG4NbP1k`, `Best Of The Big`, `null`, `compilation`, `-788918400000`, `24`
 - `6425eb31cc39`: `6fTDGzrt1Jmg`, `Jascha Heifetz`, `C`, `album`, `-788918400000`, `21`

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"  
    }  
}  
}  
}  
})
```

The screenshot shows the Studio 3T interface with the following details:

- Top Bar:** Connect, Collection, IntelliShell, SQL, Aggregate, Map-Reduce, Compare, Schema, Reschema, Tasks, Export, Import, Data Masking, SQL Migration, Users, Roles, Feedback, Go to Free.
- Message Bar:** Enjoy a 4-day full product trial. Once the trial is complete, you will be switched to Studio 3T Free or you can switch now by disabling the trial in My License (in the Help Menu).
- Left Sidebar (Open connections):** Shows a single connection to "mongodb://localhost:27017 [direct]" with the database "spotify". The "artists" collection is selected.
- Central Area:**
 - Query Bar:** Quickstart, CSV Import*, CSV Import*, artists. The connection is set to "mongodb://localhost:27017" and the database to "spotify".
 - Query Editor:** Shows the query "artists".
 - Projection: `{}`
 - Sort: `{}`
 - Limit: `50`
 - Result View:** Shows the "artists" collection with the following fields: _id, name, popularity, followers.

_id	name	popularity	followers
6425ed161cc39	50 Cent	3q7HBObvC0L8	86
6425ed161cc39	Akon	0z4gvV4jJl3w	84
6425ed161cc39	Baker St.	0k4UL5tOqUH	42
6425ed161cc39	Big Brown	05C79U7QWKC	23
6425ed161cc39	Big Daddy Kane	6fUYTSVDjuVr2F	52
6425ed161cc39	Bill Evans Trio	3VEG8pJFIMi4	58
6425ed161cc39	Busta Rhymes	1YIEctQvBQ8x	77
6425ed161cc39	Carla Anderson	1TBqlBISPOsseeG	43
6425ed161cc39	Craig Mack	4ak4juteQRgEx	55
6425ed161cc39	D12	5Qi4b7a8Coao	68
6425ed161cc39	DMX	1HmW5zC5NqW	83
6425ed161cc39	Doe Boy Philly	3WlsLMFSTuIa	23
6425ed161cc39	Dr. Dre	6DPYq5kWVQ	83
6425ed161cc39	Eminem	7dGJ4apcd2V8k	94
6425ed161cc39	Eric K. Sermon	2VXOb9LDivMmk	54
6425ed161cc39	Figg Moid	1nr44rcCUR2sl	49
6425ed161cc39	G-Unit	6evKD5WJJOSE	66
6425ed161cc39	Geoffrey Palmer	74wok4cqxpCxt	39
6425ed161cc39	Hittman	3np20YRZ4g4E	63
6425ed161cc39	J. Ro	3lbOswwGzRkyI	45
6425ed161cc39	Jewell	05cqfWtssCo3	54
6425ed161cc39	Jurat Stanki Trio	6Jup6aQGeBuer	48
6425ed161cc39	Keith Jarrett Trio	3OJG2gb1ZoC3	52
6425ed161cc39	Kurupt	6NyJfHApFjH	67
6425ed161cc39	Lloyd Banks	3vDUJHQtT3F	66
 - Operations:**
 - Import from CSV: /Users/sheyasakpal/Desktop/Spotify/artists.csv → spotify
 - Total time: 0:00:00.06566
 - Import finished
 - 1,066,031 document(s) imported.
 - Total time: 0:00:00.06566
 - Import from CSV: /Users/sheyasakpal/Desktop/Spotify/artists.csv → spotify
 - Total time: 0:00:00.06566
 - Import finished
 - 1,066,031 document(s) imported.
 - Total time: 0:00:00.06566

3. Collection audio features

```
db.createCollection("audio_features", {
```

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

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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)

_id	name
6425eeb01cc39	detroit hip hop
6425eeb01cc39	funk
6425eeb01cc39	gangster rap
6425eeb01cc39	hardcore hip hop
6425eeb01cc39	hip hop
6425eeb01cc39	pop rap
6425eeb01cc39	rap
6425eeb01cc39	west coast rap
6425eeb01cc39	hip pop
6425eeb01cc39	chilena
6425eeb01cc39	turmalism
6425eeb01cc39	east coast hip hop
6425eeb01cc39	south carolina hip hop
6425eeb01cc39	queens hip hop
6425eeb01cc39	filter house
6425eeb01cc39	conscious hip hop
6425eeb01cc39	old school hip hop
6425eeb01cc39	southern hip hop
6425eeb01cc39	anarcho-punk
6425eeb01cc39	trap
6425eeb01cc39	modern blues rock
6425eeb01cc39	modern folk rock
6425eeb01cc39	post-teen pop
6425eeb01cc39	new orleans rap
6425eeb01cc39	electro
6425eeb01cc39	virginia hip hop

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"
                }
            }
        }
    }
})
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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"
                }
            }
        }
    }
})
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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

```
}
```

```
}
```

```
}
```

```
}
```

_id	album_id	track_id
6425ef301cc39	6os2Mv580YnQ	3InfrfGL69u2M
6425ef301cc39	6os2Mv580YnQ	4DjkpUrpWIMFc
6425ef301cc39	6os2Mv580YnQ	4RDt3fT3V0LJ
6425ef301cc39	6os2Mv580YnQ	1Dfw0z2apF0w
6425ef301cc39	6os2Mv580YnQ	46JY9OYPi4hJ7
6425ef301cc39	6os2Mv580YnQ	63A7Q2jgxwF
6425ef301cc39	6os2Mv580YnQ	7a6zaB0ya7m
6425ef301cc39	6os2Mv580YnQ	55NmWifubUT9
6425ef301cc39	6os2Mv580YnQ	62KpKErs2Eyhf
6425ef301cc39	6os2Mv580YnQ	2AW0gskypXnI
6425ef301cc39	6os2Mv580YnQ	34quhtd2fSc4
6425ef301cc39	6os2Mv580YnQ	02FzbIbtqElixC
6425ef301cc39	6os2Mv580YnQ	5CMlM2f8hkm
6425ef301cc39	6os2Mv580YnQ	4LbubJ3wc4EW
6425ef301cc39	6os2Mv580YnQ	35iskz2BwRDbv
6425ef301cc39	6os2Mv580YnQ	0sj3QeaCpacCb
6425ef301cc39	6os2Mv580YnQ	1HAK7hSD1szW
6425ef301cc39	6os2Mv580YnQ	0lgEx4vkZP1y9
6425ef301cc39	6os2Mv580YnQ	67Wjz5009W
6425ef301cc39	6os2Mv580YnQ	50hE4kDxwXw
6425ef301cc39	6os2Mv580YnQ	11m7iaUgnOK
6425ef301cc39	6os2Mv580YnQ	4HymmG8uHL2
6425ef301cc39	6os2Mv580YnQ	7FznWeEpRxM
6425ef301cc39	6os2Mv580YnQ	50lCylqlyxAOV
6425ef301cc39	6os2Mv580YnQ	0cSOAtfUEudE
6425ef301cc39	5XKN1FG707L	OhbkKFUjm7Z05

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"
```

```
                },
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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

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

_id	genre_id	artist_id
6425f06c1cc39f	detroit hip hop	4tuJQJicOnuZRL
6425f06c1cc39f	g funk	4tuJQJicOnuZRL
6425f06c1cc39f	gangster rap	4tuJQJicOnuZRL
6425f06c1cc39f	hardcore hip hop	4tuJQJicOnuZRL
6425f06c1cc39f	hip hop	4tuJQJicOnuZRL
6425f06c1cc39f	pop rap	4tuJQJicOnuZRL
6425f06c1cc39f	rap	4tuJQJicOnuZRL
6425f06c1cc39f	west coast rap	4tuJQJicOnuZRL
6425f06c1cc39f	gangster rap	2VX0o9LDVmKI
6425f06c1cc39f	hardcore hip hop	2VX0o9LDVmKI
6425f06c1cc39f	hip hop	2VX0o9LDVmKI
6425f06c1cc39f	hip pop	2VX0o9LDVmKI
6425f06c1cc39f	rap	2VX0o9LDVmKI
6425f06c1cc39f	turntablism	2VX0o9LDVmKI
6425f06c1cc39f	gangster rap	4ak4juteQOrGx
6425f06c1cc39f	hardcore hip hop	4ak4juteQOrGx
6425f06c1cc39f	hip hop	4ak4juteQOrGx
6425f06c1cc39f	rap	4ak4juteQOrGx
6425f06c1cc39f	detroit hip hop	7dGJ04pcD2V6k
6425f06c1cc39f	hip hop	7dGJ04pcD2V6k
6425f06c1cc39f	rap	7dGJ04pcD2V6k
6425f06c1cc39f	g funk	6DPYiyq5kWVQ
6425f06c1cc39f	gangster rap	6DPYiyq5kWVQ
6425f06c1cc39f	hip hop	6DPYiyq5kWVQ
6425f06c1cc39f	rap	6DPYiyq5kWVQ
6425f06c1cc39f	west coast rap	6DPYiyq5kWVQ

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"  
        }  
    }  
}  
})
```

The screenshot shows the MongoDB Compass application interface. The left sidebar displays the database structure for 'spotify' with collections like 'albums', 'artists', 'audio_features', 'genres', 'r_albums_artists', 'r_albums_tracks', 'r_artist_genre', and 'r_track_artist'. The 'r_track_artist' collection is currently selected. The main pane shows a table of results for the query '_id': '6425f0bb1cc39'. The table has columns: '_id', 'track_id', and 'artist_id'. The results list numerous document entries, each containing a unique '_id', a track identifier, and an artist identifier. The bottom of the screen shows the status bar with '1 document selected' and the bottom right corner shows the timestamp '00:00:00.002'.

9. Collection tracks

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

validator: {

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

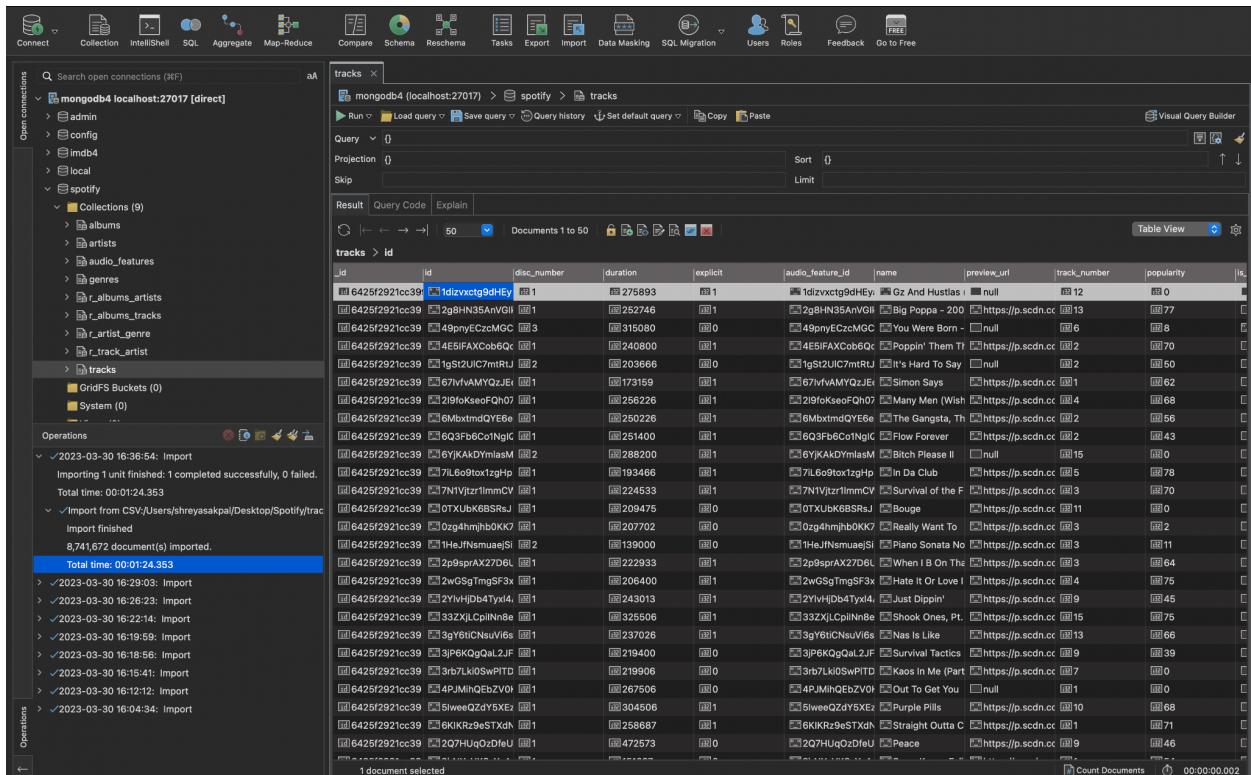
Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

```
$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)

The screenshot shows a DBeaver interface with two panes. The top pane is the SQL editor containing the following query:

```
-- 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;
```

The bottom pane is the Output pane titled "main.artists" showing a table with 13 rows of data:

	name	followers
1	Creedence Clearwater Revival	4181740
2	Etta James	1165157
3	X Ambassadors	2407676
4	Outkast	1619951
5	Nirvana	12442956
6	Ariana Grande	60762233
7	The Beatles	19129092
8	Rihanna	42093194
9	DJ Snake	7299939
10	Taylor Swift	38455481
11	Ariana Grande	60762233
12	Jay Sean	1292771
13	One Direction	20795285

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)

The screenshot shows the pgAdmin 4 interface. The Database Explorer on the left lists databases, schemas, and tables. The SQL console at the top has a query window with the following code:

```
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);

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
End t.explicit = 0;
```

The Services tab at the bottom shows the execution time: 29s 578ms.

name	followers
Etta James	1165157
The Beatles	19129992

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)

The screenshot shows a PostgreSQL database interface. On the left, there's a tree view of databases (assignment_2@localhost), tables (main.tracks, main.r_track_artist, main.r_artist_genre), and other objects. In the center, a query editor window displays a SQL script. The script includes comments and several queries related to tracks and genres. Below the editor is a results table showing a single row: 'Can't Stop Loving - Original Mix' with a duration of 5655335 ms. The bottom status bar indicates the query took 5m 56s 901ms to execute.

```
and t.explicit = 0;
-- 3. Longest duration of tracks, whose genre name like %jazz%, which belongs to an album with type ....
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%';
-- 4. genres that have the most number of songs and sort by descending order and display only top 5, with the audio
```

name	"max(main.tracks.duration)"
Can't Stop Loving - Original Mix	5655335

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_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)

The screenshot shows the pgAdmin interface with a database named 'spotify'. In the 'Console' tab, a query is being run:

```
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 );
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%';
```

The 'Output' tab shows the result of the query:

name	max(main.tracks.duration)
Can't Stop Loving - Original Mix	5655335

The 'Services' panel on the left shows the database connection status.

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)

The screenshot shows the Database Explorer interface with several tabs open. The main tab displays a SQL query:

```
-- 4. 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;
```

The Services tab shows the execution time for each query, with the last query taking 24m 53s.

id	avg_loudness	num_songs
1	-22.267453537693612	471010
2	-21.796343203469533	355748
3	-12.689557179049775	199006
4	-21.372028299185644	170932
5	-9.109785994617498	162060

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)

The screenshot shows a PostgreSQL database interface with multiple tabs open. The top tab is 'console [spotify]'. The code being run is:

```
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 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 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);

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;
```

The results of the query are displayed in the 'Result' tab:

id	avg_loudness	num_songs
1	-22.26745537693612	471010
2	-21.776343283469533	355748
3	-12.689557179849775	199086
4	-21.372628299185644	178932
5	-9.19785994617498	162060

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)

The screenshot shows the pgAdmin interface. In the Database Explorer, under the 'spotify' database, the 'genres' table is selected. In the Services tab, a query is run:

```
--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;
```

The results show the top 27 genres and their track counts:

id	tracks
1	471060
2	355779
3	199888
4	178950
5	162660
6	125852
7	124985
8	121505
9	118975
10	115960
11	109518
12	106118
13	103797
14	103667
15	101691
16	97977
17	96233
18	91378
19	90483
20	89215
21	85619
22	85454
23	83202
24	82565
25	81947
26	81599
27	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)

The screenshot shows the pgAdmin interface with a database named 'spotify'. In the top-left pane, under 'Database Explorer', there are several connections and their execution times: 'postgres@localhost 1 of 6' (1ms), 'spotify 1 of 4' (1ms), 'console_1 70 ms', 'audio_features 70 ms', and 'console 15 s 421 ms'. The main pane displays two queries. The first query creates indexes for genre, artist_genre, artist_id, albums_artists, artists_album_id, and albums_tracks. The second query retrieves the names of all genres and the number of tracks associated with each genre, sorted by the number of tracks in descending order. The results table shows 31 rows of genre names and their corresponding track counts.

id	tracks
1	471060
2	355779
3	199008
4	170950
5	162660
6	125852
7	124985
8	121505
9	118975
10	115960
11	109518
12	106118
13	103797
14	103667
15	101691
16	97977
17	96233
18	91378
19	90485
20	89215
21	85619
22	85454
23	83282
24	82565
25	81947
26	81599
27	80889
28	75157
29	74146
30	74004
31	73838

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)

The screenshot shows a database interface with a code editor and a results viewer. The code editor contains a SQL query to select tracks longer than 5 minutes, joining multiple tables: tracks, r_albums_tracks, albums, r_albums_artists, artists, r_artist_genre, and genres. The results viewer displays a table with columns: track_name, album_name, artist_name, and genre_name. The data shows 11 rows of songs by 'Shook Ones, Pt. II' from 'The Infamous' album, performed by 'Mobb Deep', categorized under various genres like 'east coast hip hop', 'gangster rap', etc.

track_name	album_name	artist_name	genre_name
Shook Ones, Pt. II	The Infamous	Mobb Deep	east coast hip hop
Shook Ones, Pt. II	The Infamous	Mobb Deep	gangster rap
Shook Ones, Pt. II	The Infamous	Mobb Deep	hardcore hip hop
Shook Ones, Pt. II	The Infamous	Mobb Deep	hip hop
Shook Ones, Pt. II	The Infamous	Mobb Deep	queens hip hop
Shook Ones, Pt. II	The Infamous	Mobb Deep	rap
Purple Pills	Devils Night	D12	detroit hip hop
Purple Pills	Devils Night	D12	gangster rap
Purple Pills	Devils Night	D12	hardcore hip hop
Purple Pills	Devils Night	D12	hip hop
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)

The screenshot shows a PostgreSQL database interface with several tabs open. The top tab is 'console [spotify]'. The left sidebar shows a tree view of the database schema under 'postres@localhost'. Under the 'spotify' schema, there are tables for 'albums', 'artists', 'audio_features', 'genres', 'tracks', and 'sequences'. The 'genres' table is currently selected. The main area displays two code snippets. The first snippet is a series of SQL commands used to create various indexes on the 'tracks', 'r_albums_tracks', 'r_albums', 'r_albums_artists', 'r_albums_artists_artists', 'r_artist_genre', 'r_artist_genre_genre_id_idx', and 'genres_id_idx' tables. The second snippet is a query result table titled 'Output' with columns 'track_name', 'album_name', 'artist_name', and 'genre_name'. The table contains 26 rows of data, each representing a song with its album, artist, and genre information.

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**

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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**

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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.