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

BIGDATA PROJECT PHASE - III
GROUP 34 - (SPOTIFY 12-M SONGS DATASET)

CLEANING AND INTEGRATION OF DATA

```
def pre_artists():
    r_artists = 'C:\\Users\\anush\\Documents\\CSCI-620 Big ' \
                'Data\\Movies\\title.basics.tsv '
    artists = pd.read_table(r_artists)
    artists = artists.drop_duplicates()  # drop duplicates
    artists = artists.dropna()  # filter null values
    artists.to_csv('C:\\Users\\anush\\Documents\\CSCI-620 Big '
                   'Data\\Assignment 2\\director.csv', index=False,
                   header=False)
    print('Actor table preprocessed successfully')
```

```
def pre_albums():
    r_albums = 'C:\\Users\\anush\\Documents\\CSCI-620 Big ' \
               'Data\\Movies\\title.basics.tsv '
    albums = pd.read_table(r_albums)
    albums = albums.drop(['album_group'], axis=1)  # drop columns
    albums = albums.drop_duplicates()  # drop duplicates
    albums = albums.dropna()  # filter null values
    albums.to_csv('C:\\Users\\anush\\Documents\\CSCI-620 Big '
                  'Data\\Assignment 2\\director.csv', index=False,
                  header=False)
    print('Albums table preprocessed successfully')
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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

```
def pre_tracks():
    r_tracks = 'C:\\Users\\anush\\Documents\\CSCI-620 Big ' \
               'Data\\Movies\\title.basics.tsv '
    tracks = pd.read_table(r_tracks)
    tracks = tracks.drop(['disc_number', 'preview_url'], axis=1)  # drop columns
    tracks = tracks.drop_duplicates()  # drop duplicates
    tracks = tracks.dropna()  # filter null values
    tracks.to_csv('C:\\Users\\anush\\Documents\\CSCI-620 Big '
                  'Data\\Assignment 2\\director.csv', index=False,
                  header=False)
    print('Tracks table preprocessed successfully')
```

```
def pre_audio_features():
    r_audio_features = 'C:\\Users\\anush\\Documents\\CSCI-620 Big ' \
                       'Data\\Movies\\title.basics.tsv '
    audio_features = pd.read_table(r_audio_features)
    # drop columns
    audio_features = audio_features.drop(['analysis_url', 'key', 'mode',
                                          'speechiness', 'tempo',
                                          'time_signature', 'valence'], axis=1)
    audio_features = audio_features.drop_duplicates()  # drop duplicates
    audio_features = audio_features.dropna()  # filter null values
    audio_features.to_csv('C:\\Users\\anush\\Documents\\CSCI-620 Big '
                          'Data\\Assignment 2\\director.csv', index=False,
                          header=False)
    print('Audio Features table preprocessed successfully')
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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

```
def pre_genres():
    r_genres = 'C:\\Users\\anush\\Documents\\CSCI-620 Big ' \
               'Data\\Movies\\title.basics.tsv '
    genres = pd.read_table(r_genres)
    genres = genres.drop_duplicates()  # drop duplicates
    genres = genres.dropna()  # filter null values
    genres.to_csv('C:\\Users\\anush\\Documents\\CSCI-620 Big '
                  'Data\\Assignment 2\\director.csv', index=False,
                  header=False)
    print('Genres table preprocessed successfully')
```

The program defines several functions, each of which preprocesses a specific table. These functions perform the following tasks:

- pre_artists(): Reads the TSV file for the artists table, drops any duplicate rows, and removes any rows with missing values. The resulting cleaned data is saved to a new CSV file.
- pre_albums(): Reads the TSV file for the albums table, drops the album_group column, drops any duplicate rows, and removes any rows with missing values. The resulting cleaned data is saved to a new CSV file.
- pre_tracks(): Reads the TSV file for the tracks table, drops several columns, drops any duplicate rows, and removes any rows with missing values. The resulting cleaned data is saved to a new CSV file.
- pre_audio_features(): Reads the TSV file for the audio features table, drops several columns, drops any duplicate rows, and removes any rows with missing values. The resulting cleaned data is saved to a new CSV file.
- pre_genres(): Reads the TSV file for the genres table, drops any duplicate rows, and removes any rows with missing values. The resulting cleaned data is saved to a new CSV file.

The main function calls each of these preprocessing functions in turn. The cleaned data is saved to CSV files, which can be used for further analysis.
Overall, the program uses the pandas library to read and manipulate the data, dropping duplicate rows and removing any rows with missing values. The resulting cleaned data is saved to new CSV files, which can be used for further analysis.

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

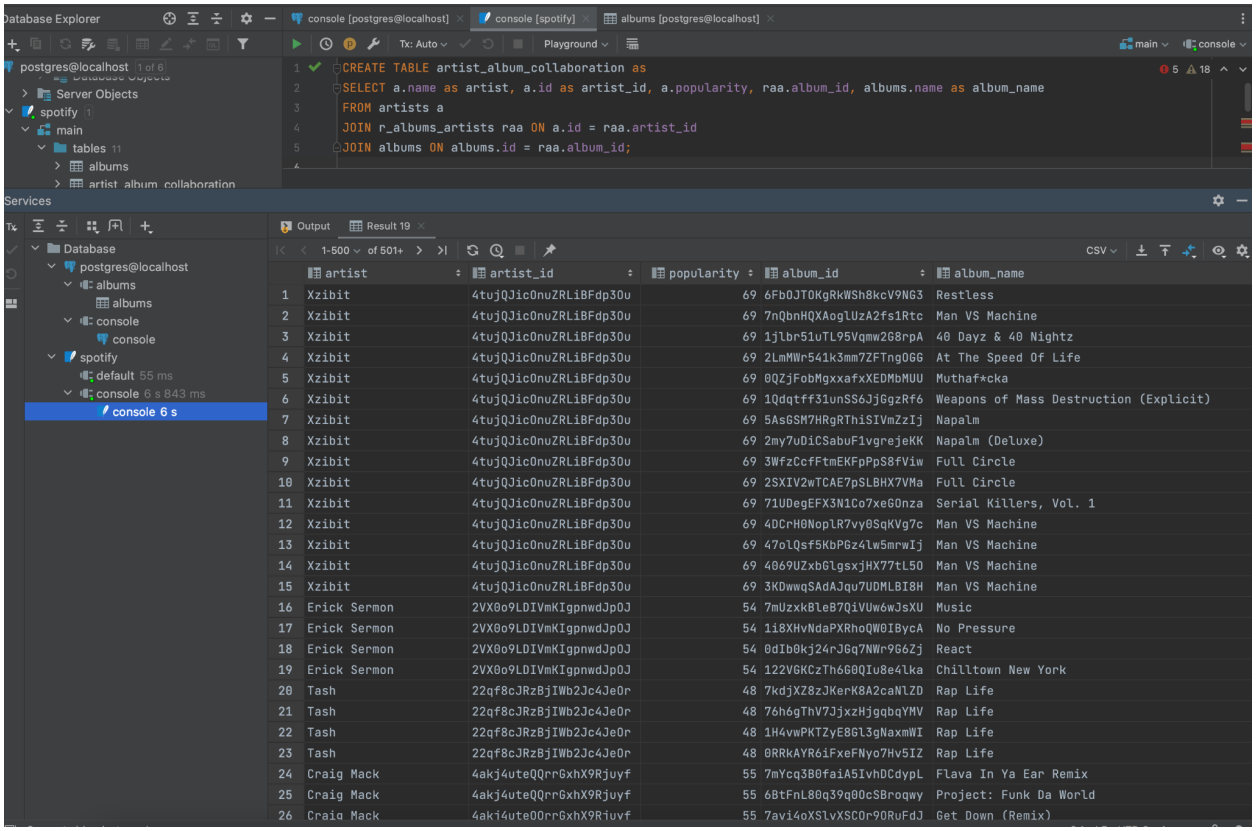
Anushka Churi (ac2224)
Meeti Dixit (md2335)
Neha Kulkarni (nk4349)
Shreya Sakpal (ss7807)

LATTICE CREATION USING RELATIONAL MODEL

First, we create a table by joining our existing tables in order to perform Itemset Mining. Here, we have created a table entitled “Artist Album Collaboration” where we display the artists, their respective ids, their popularity and the albums they have created along with the album ids.

TABLE ARTIST ALBUM COLLABORATION

```
CREATE TABLE artist_album_collaboration as
SELECT a.name as artist, a.id as artist_id, a.popularity, raa.album_id,
albums.name as album_name
FROM artists a
JOIN r_albums_artists raa ON a.id = raa.artist_id
JOIN albums ON albums.id = raa.album_id;
```



	artist	artist_id	popularity	album_id	album_name
1	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	6FbOJTOKgRkWSh8kcV9NG3	Restless
2	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	7nQbnHQXAoglUzA2fs1Rtc	Man VS Machine
3	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	1jlbr51uTL95Vqmw2G8rpA	40 Dayz & 40 Nightz
4	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	2LmMWr541k3mm7ZFTngOGG	At The Speed Of Life
5	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	0QZjFobMgxxafxXEDMbMUU	Muthaf*cka
6	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	1Qdqtff31unSS6JjGgzRf6	Weapons of Mass Destruction (Explicit)
7	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	5AsGSM7HRgRThiSIVmZzIj	Napalm
8	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	2my7uDiCSabuF1vgrejeKK	Napalm (Deluxe)
9	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	3WfzCcfFtmEKFpPpS8fViw	Full Circle
10	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	2SXIV2wTCAE7pSLBHX7VMa	Full Circle
11	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	71UDegEFX3N1Co7xeGOnza	Serial Killers, Vol. 1
12	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	4DCrH0NoplR7vy0SqKVg7c	Man VS Machine
13	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	47olQsf5KbPGz4lw5mrwIj	Man VS Machine
14	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	4069UZxbGlgsxjHX77tL5O	Man VS Machine
15	Xzibit	4tujQJicOnuZRLiBFdp3Ou	69	3KDwwqSAdAJqu7UDMLBI8H	Man VS Machine
16	Erick Sermon	2VX0o9LDIVmKIgpnwdJpOJ	54	7mUzxkBleB7QiVUw6wJsXU	Music
17	Erick Sermon	2VX0o9LDIVmKIgpnwdJpOJ	54	1i8XHvNdaPXRhoQW0IBycA	No Pressure
18	Erick Sermon	2VX0o9LDIVmKIgpnwdJpOJ	54	0dIb0kj24rJGq7NWr9G6Zj	React
19	Erick Sermon	2VX0o9LDIVmKIgpnwdJpOJ	54	122VGKCzTh6G0QIu8e4lka	Chilltown New York
20	Tash	22qf8cJRzBjIWb2Jc4JeOr	48	7kdjXZ8zJKerK8A2caNlZD	Rap Life
21	Tash	22qf8cJRzBjIWb2Jc4JeOr	48	76h6gThV7JjxzHjgqbqYMV	Rap Life
22	Tash	22qf8cJRzBjIWb2Jc4JeOr	48	1H4vwPKTZyE8Gl3gNaxmWI	Rap Life
23	Tash	22qf8cJRzBjIWb2Jc4JeOr	48	0RRkAYR6iFxeFNyo7Hv5IZ	Rap Life
24	Craig Mack	4akj4uteQQrrGxhX9Rjuyf	55	7mYcq3B0faiA5IvhDCdypL	Flava In Ya Ear Remix
25	Craig Mack	4akj4uteQQrrGxhX9Rjuyf	55	6BtFnL80q39q0OcSBroqwy	Project: Funk Da World
26	Craig Mack	4akj4uteQQrrGxhX9Rjuyf	55	7avi4oXSlvXSCOr9ORuFdJ	Get Down (Remix)

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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

LATTICE L1

```
create table L1 as
select artist_id as artist1, album_name, count(*) as count
from artist_album_collaboration
where popularity > 50
group by artist
having count(*) > 5;
```

The above query creates a table named "L1" using the "create table" command in SQL. The table has three columns: "artist1", "album_name", and "count". The "artist1" column contains the artist IDs, the "album_name" column contains the names of the albums created by the artists, and the "count" column contains the number of times the artist has collaborated on an album with other artists.

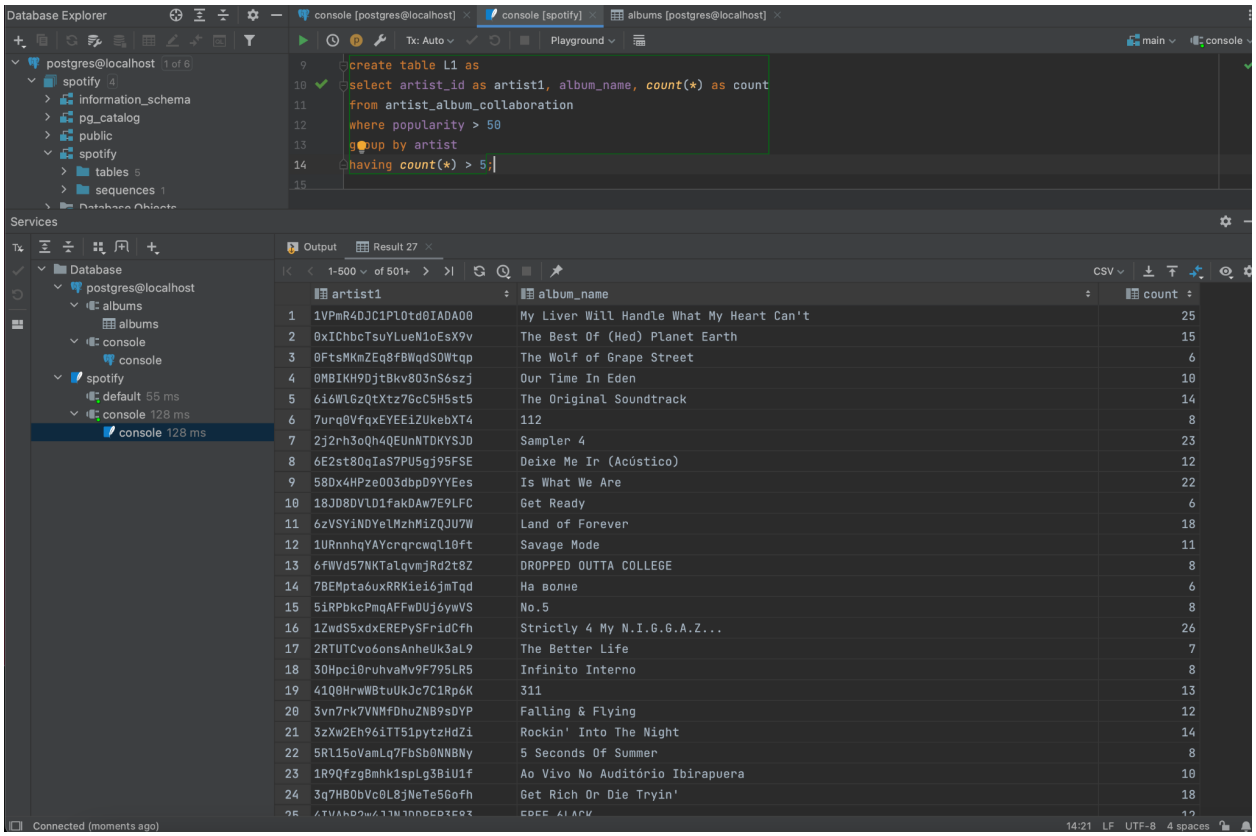
The query selects data from the "artist_album_collaboration" table where the popularity of the artist is greater than 50 and groups the data by the artist ID. **It then selects only those artists who have collaborated on at least 6 albums (count(*) > 5)** and displays the artist ID, album name, and count in the L1 table.

To create lattices using the above query, we need to perform itemset mining on the "L1" table. Itemset mining is a data mining technique that involves finding frequent itemsets in a transactional database. In this case, we can consider each album as a transaction, and the artists who collaborated on the album as items.

We can create lattices using the frequent itemsets obtained from the itemset mining process.

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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



LATTICE L2

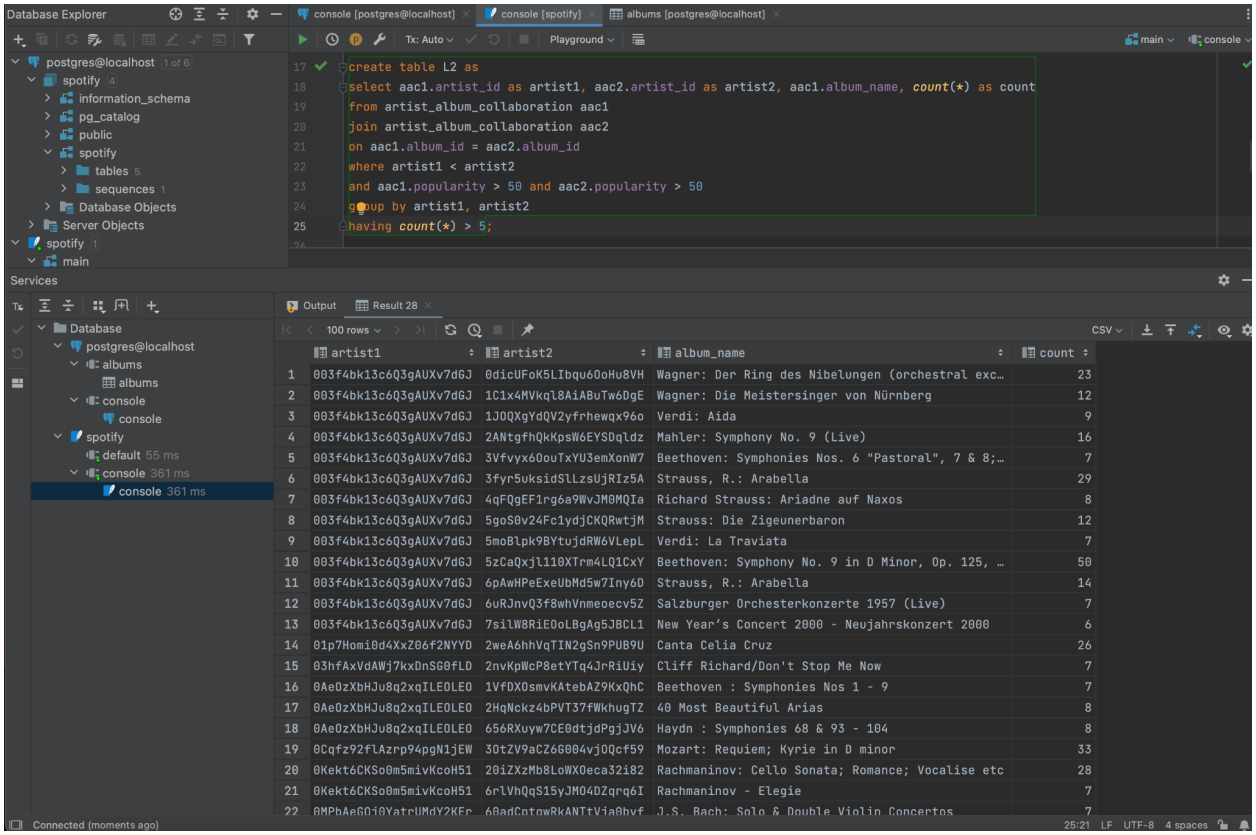
```
create table L2 as
select aac1.artist_id as artist1, aac2.artist_id as artist2, aac1.album_name,
count(*) as count
from artist_album_collaboration aac1
join artist_album_collaboration aac2
on aac1.album_id = aac2.album_id
where artist1 < artist2
and aac1.popularity > 50 and aac2.popularity > 50
group by artist1, artist2
having count(*) > 5;
```

The query selects data from the "artist_album_collaboration" table by joining the table to itself using the "join" command. The join condition is that both rows being joined have the same "album_id", which means they are both collaborations on the same album. The query also filters the results to only include collaborations between two artists with a popularity greater than 50.

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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

The resulting data is then grouped by the IDs of the two artists, and only those pairs of artists who have collaborated on at least 6 albums are selected (having count(*) > 5). The resulting data is then inserted into the "L2" table.



LATTICE L3

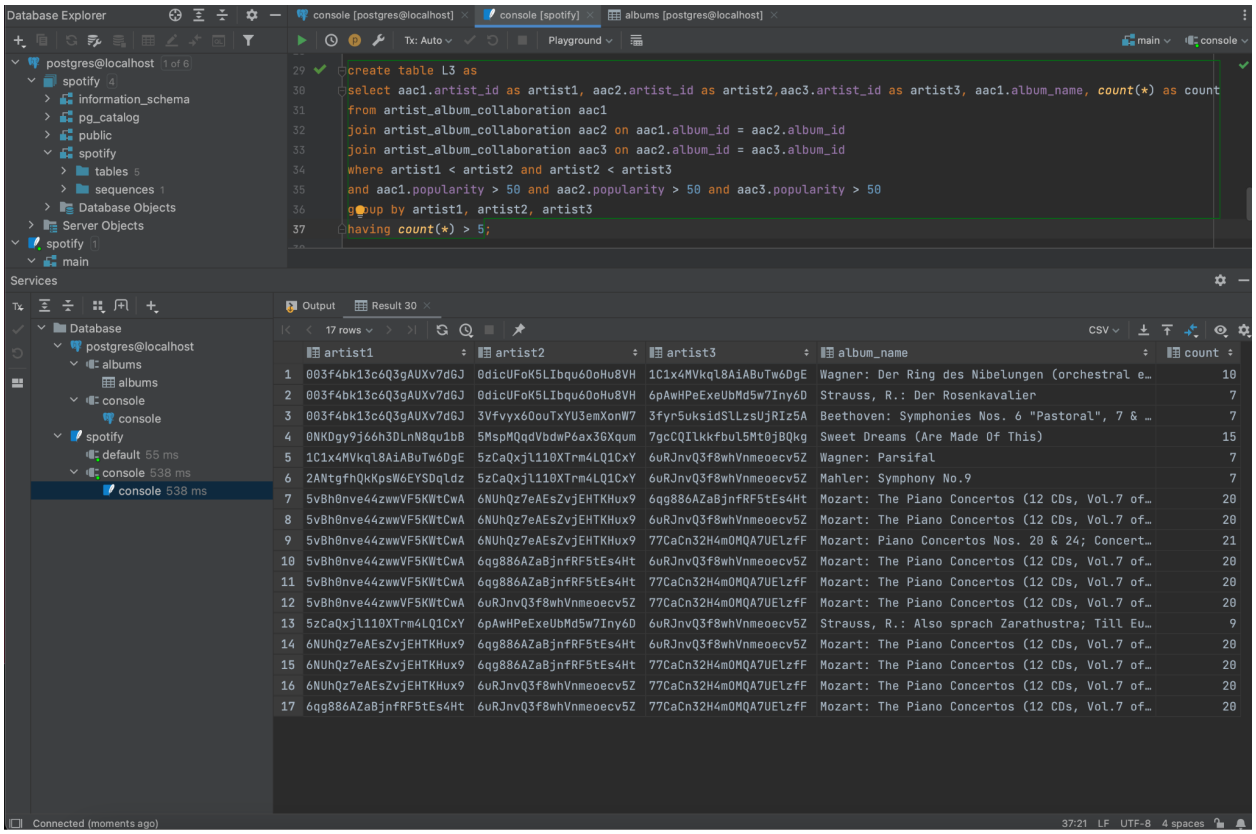
```
create table L3 as
select aac1.artist_id as artist1, aac2.artist_id as artist2,aac3.artist_id as
artist3, aac1.album_name, count(*) as count
from artist_album_collaboration aac1
join artist_album_collaboration aac2 on aac1.album_id = aac2.album_id
join artist_album_collaboration aac3 on aac2.album_id = aac3.album_id
where artist1 < artist2 and artist2 < artist3
and aac1.popularity > 50 and aac2.popularity > 50 and aac3.popularity > 50
group by artist1, artist2, artist3
having count(*) > 5;
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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

The "artist1", "artist2", and "artist3" columns from the above query contain the IDs of three collaborating artists who have collaborated on an album. The "album_name" column contains the name of the album they collaborated on, and the "count" column contains the number of times the three artists collaborated on an album.
The query selects data from the "artist_album_collaboration" table by joining it to itself three times using the "join" command. The join conditions are that each row being joined has the same "album_id", which means they are all collaborations on the same album, and that the IDs of the three artists are in increasing order.

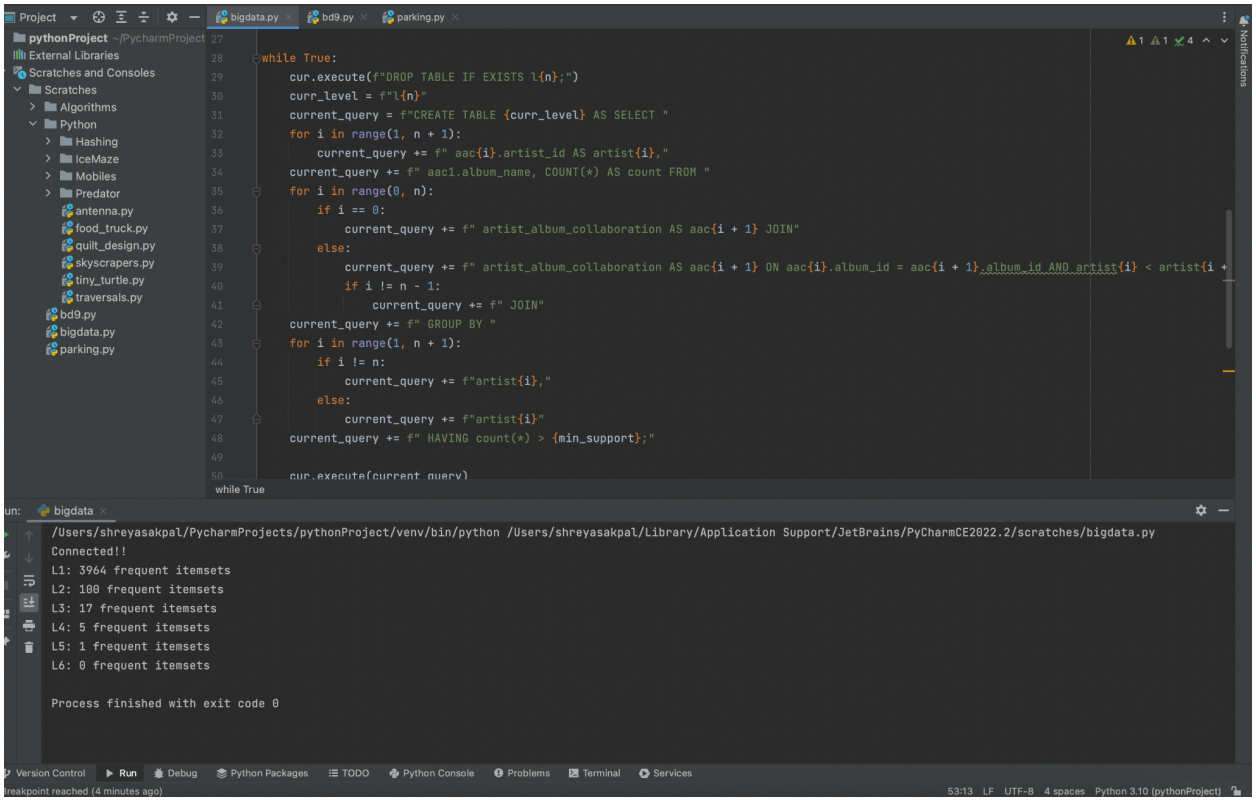
The query also filters the results to only include collaborations between three artists with a popularity greater than 50. The resulting data is then grouped by the IDs of the three artists, and only those triplets of artists who have collaborated on at least 6 albums are selected (having count(*) > 5). The resulting data is then inserted into the "L3" table.



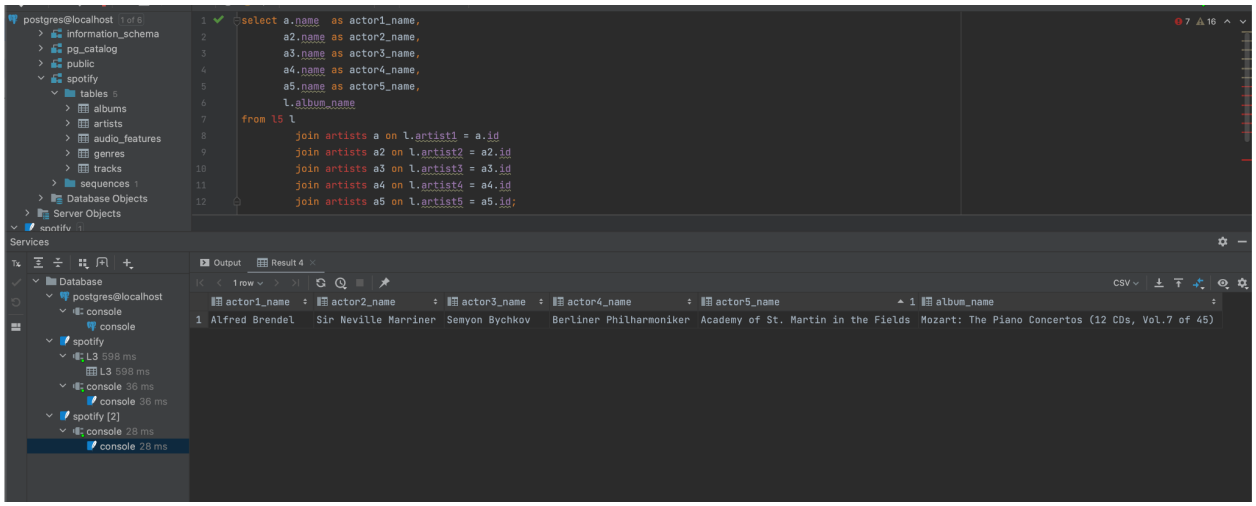
GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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

APRIORI ALGORITHM TO GENERATE LATTICES



ARTISTS IN THE LAST NON-ZERO LATTICE (L5)



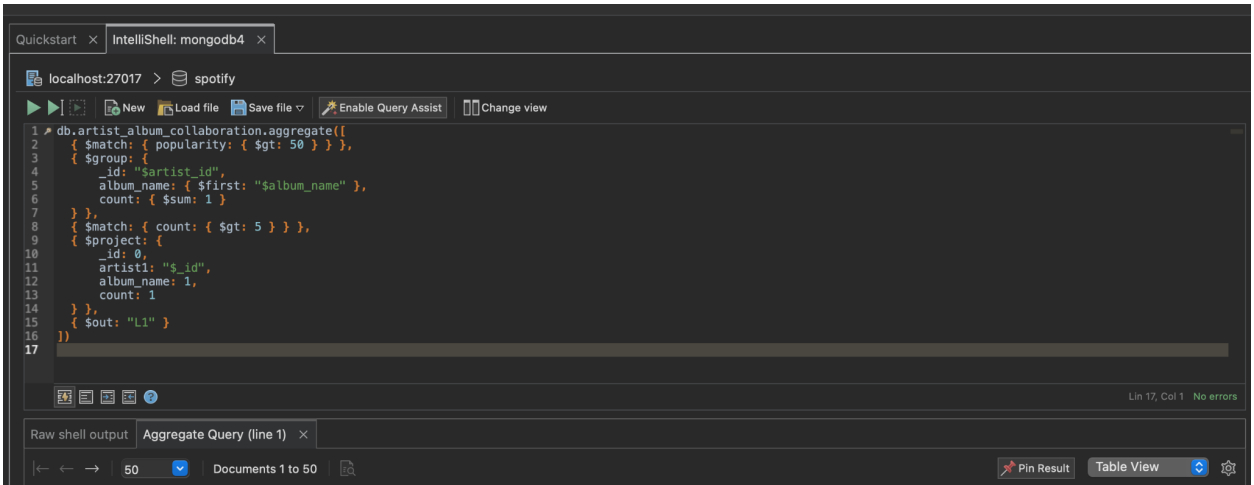
GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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

LATTICE CREATION USING DATA-ORIENTED MODEL

LATTICE L1

```
db.artist_album_collaboration.aggregate([

{ $match: { popularity: { $gt: 50 } } },

{ $group: {

_id: "$artist_id",

album_name: { $first: "$album_name" },

count: { $sum: 1 }

} },

{ $match: { count: { $gt: 5 } } },

{ $project: {

_id: 0,

artist1: "$_id",

album_name: 1,

count: 1

} },

{ $out: "L1" }

])
```



GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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

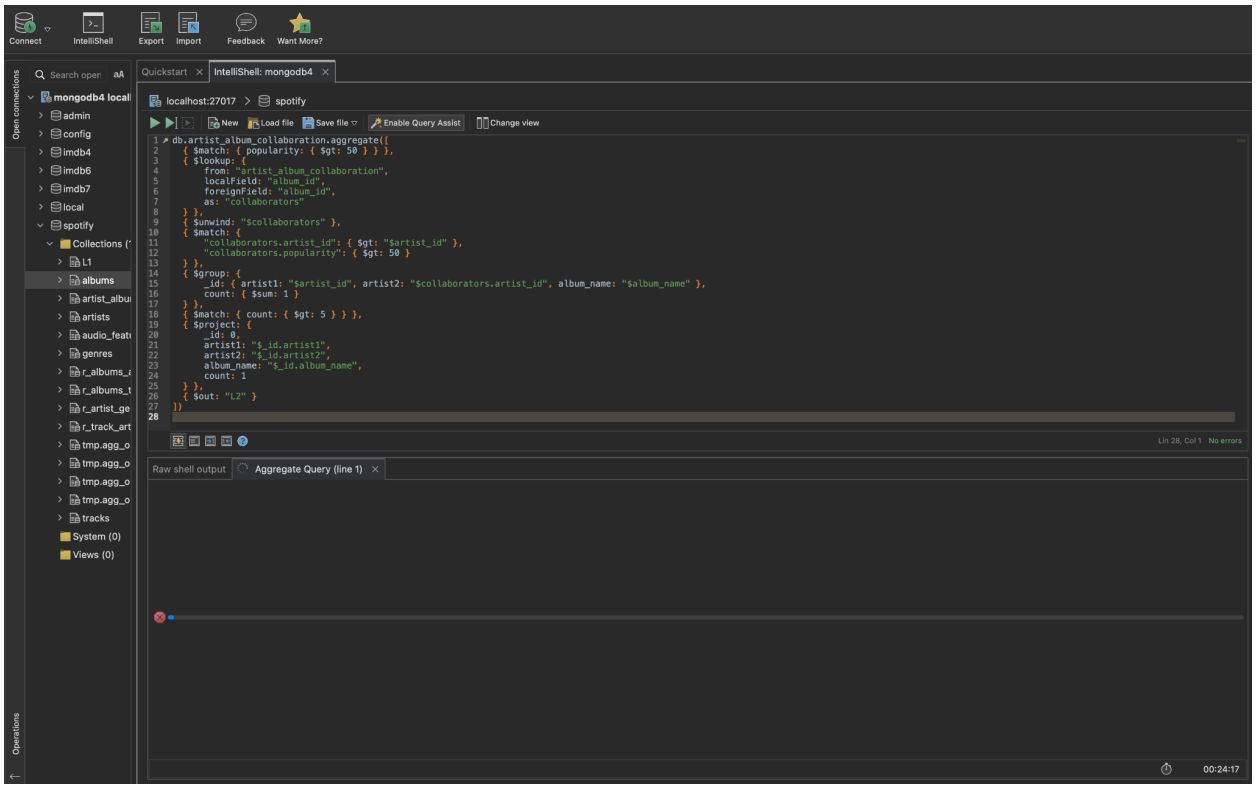
LATTICE L2

```
db.artist_album_collaboration.aggregate([

{ $match: { popularity: { $gt: 50 } } },

{ $lookup: {

from: "artist_album_collaboration",

localField: "album_id",

foreignField: "album_id",

as: "collaborators"

} },

{ $unwind: "$collaborators" },

{ $match: {

"collaborators.artist_id": { $gt: "$artist_id" },

"collaborators.popularity": { $gt: 50 }

} },

{ $group: {

_id: { artist1: "$artist_id", artist2: "$collaborators.artist_id", album_name: "$album_name" },

count: { $sum: 1 }

} },

{ $match: { count: { $gt: 5 } } },

{ $project: {

_id: 0,

artist1: "$_id.artist1",

artist2: "$_id.artist2",

album_name: "$_id.album_name",
```

GROUP_34 (SPOTIFY 12-M SONGS DATASET)

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

```
count: 1

} },

{ $out: "L2" }

])
```



In the above lattice creation, we see that the query takes a lot of time for execution. From this we observe that itemset mining in a relational model is typically faster than in a document-oriented model due to the relational model's emphasis on structured data and built-in support for join operations. This is because itemset mining often requires comparing items across different records or collections, and relational databases are designed to efficiently execute these types of operations.

On the other hand, document-oriented databases like MongoDB are optimized for flexibility and scalability, rather than structured data and join operations. This means that executing itemset mining operations in MongoDB can be more time-consuming and

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

resource-intensive than in a relational database. MongoDB can struggle with large datasets, especially when the data has complex nested structures or requires significant preprocessing to make it suitable for mining.

One reason MongoDB may take longer to execute lattices than a relational database is because MongoDB requires more complex aggregation pipelines to group and aggregate data. These pipelines may involve multiple stages of filtering, sorting, and grouping, which can be time-consuming to execute. Additionally, MongoDB may require more memory and disk space to store and process the data than a relational database, which can slow down the mining process.

Overall, while MongoDB and other document-oriented databases can be used for itemset mining, they may not be the most efficient choice for this task compared to a relational database. However, the choice of database depends on the specific needs and requirements of the mining task, and it is always important to carefully evaluate and compare the performance of different databases before making a final decision.