HW 1 – Apache Cassandra

Matthew koton: 806614

Nofar Yungman: 205985815

https://github.com/nofaryung/BDP_IDC_2024

Dataset selection:

https://www.kaggle.com/datasets/tonygordonjr/spotify-dataset-2023/data?select=spotify_artist_data_2023.csv

We have selected the Spotify Dataset 2023 for analysis and exploration. This dataset provides a comprehensive view of Spotify's music ecosystem, offering valuable insights into albums, artists, tracks, and features. The motivation behind choosing this dataset stems from its well-structured schema, coupled with its creation process's transparency, making it an ideal choice for deriving meaningful insights and conducting in-depth analyses.

The Spotify Dataset 2023 comprises several CSV files, each focusing on specific aspects of the music platform. The key files include:

spotify-albums_data_2023.csv:Spotify catalogue information for multiple albums identified by their Spotify IDs.

spotify_artist_data_2023.csv: Offers insights into artists, including their names, genres, and popularity.

spotify_features_data_2023.csv: Encompasses features of tracks, such as danceability, energy, and instrumentalness, contributing to a more detailed analysis of music attributes.

spotify_tracks_data_2023.csv: Contains data on individual tracks, including track names, artists, and associated albums.d

Popularity and Release year table:

database schema and rationale for the design:

With this table we aim to extract meaningful insight from the data about popularity of albums throughout different years. Due to this goal and the decentralised nature of apache cassandra we designed our table to have a primary key of album popularity and release year. This allows for efficient, fast and scalable data access with respect to the aim stated above.

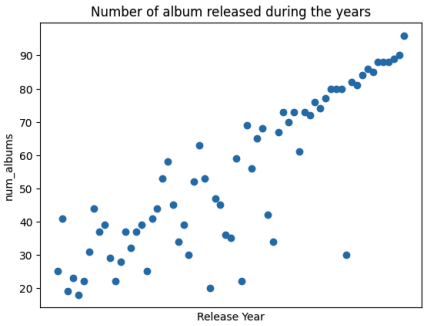
```
result_set = session.execute("""
SELECT release_year, AVG(album_popularity) AS avg_popularity, COUNT(*) AS num_albums
FROM release_year_album_popularity
GROUP BY release_year;""")
rows = list(result_set)
```

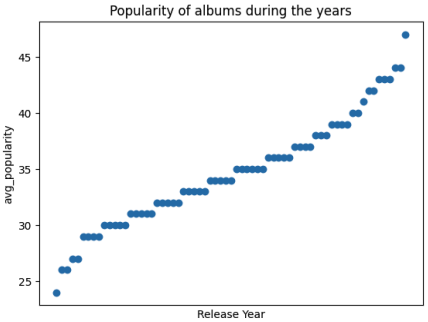
Above is an example of a query we used to gain insight into our data. Note that this query is only possible due to the design of our table having a partition key of album_popularity. This further shows the rationale behind our table design.

Insights:

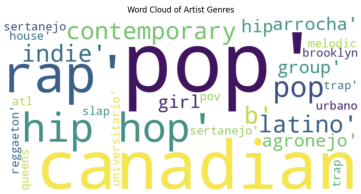
We can see that during the year the amount of albums that are being released has increased and accordingly the popularity of albums. There is a clear trend in the amount of music being released and

the amount of people listening to this music. From the graphs below we can see every year more albums are released than in the previous years and these albums are more popular than the albums released in previous years





We tried to explore what are the ruling genres and the least popular genres in the most popular year in music which is 2023 and here are our findings:



Popularity table:

We created an additional table which holds all the tracks, their popularities and a variety of features describing the tracks such as “dancibility”. We partitioned this table by track_popularity, track_id. With this table we aimed to find trends in tracks which are popular and the tracks which are not. This goal informed our table design. We also included all the track features in the same table in order to minimise the need for “join” operations.

The following python functions where used to easily query this table:

```
def find_average_of_property_with_popularity_range(property, start, end):
    avgs = []
    for popularity in range(start, end + 1):
        statement = f"SELECT AVG({property}) FROM track_info_popularity WHERE track_popularity = '{popularity}';"
        avgs.append(session.execute(statement).one()[0])
    return(avgs)
```

When exploring the data through the above queries we found that tracks which were popular and tracks which were not tended to have average similar feature values. There was no drastic difference in for example the “dancibility” of tracks which are popular and those which are not. This shows the diversity of music and the variety of songs which are popular today.

Challenges faced and how they were overcome:

At first we faced the challenge of loading large amounts of unsorted data into the tables. With our first data ingestion implementations the load time was incredibly long. We therefore decided to cut our data down in size. This allowed us to load a quarter of the csv into the table in about 7 minutes. This load time was too long and we were also not satisfied with reducing our data. We therefore further optimised our ingestion function and were able to ingest the full CSV in only a few minutes. We did this by first sorting our dataframe by the partition key and then inserting into the table using batches. We found that sorting the dataframe first led to a large improvement in the time it took to load the data into the table.