Database normalization course project presentation

By Otabek Eshpo'latov, 2nd year student at ITPU
Aprel, 2024

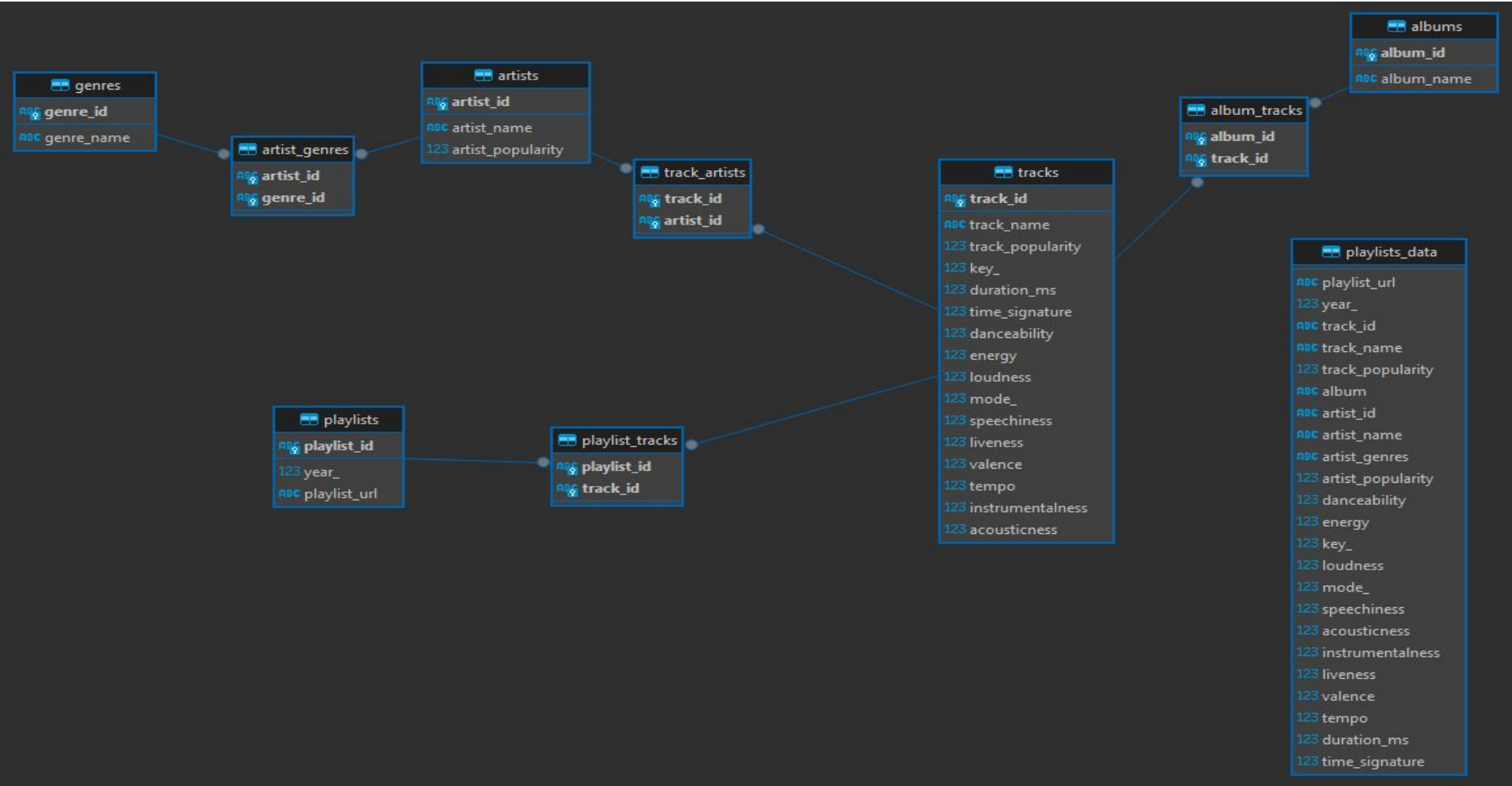
Brief introduction to the business area problem

In the course project, we explored database denormalization steps using a real-world dataset for the top Spotify hot playlist between 2010 and 2022.

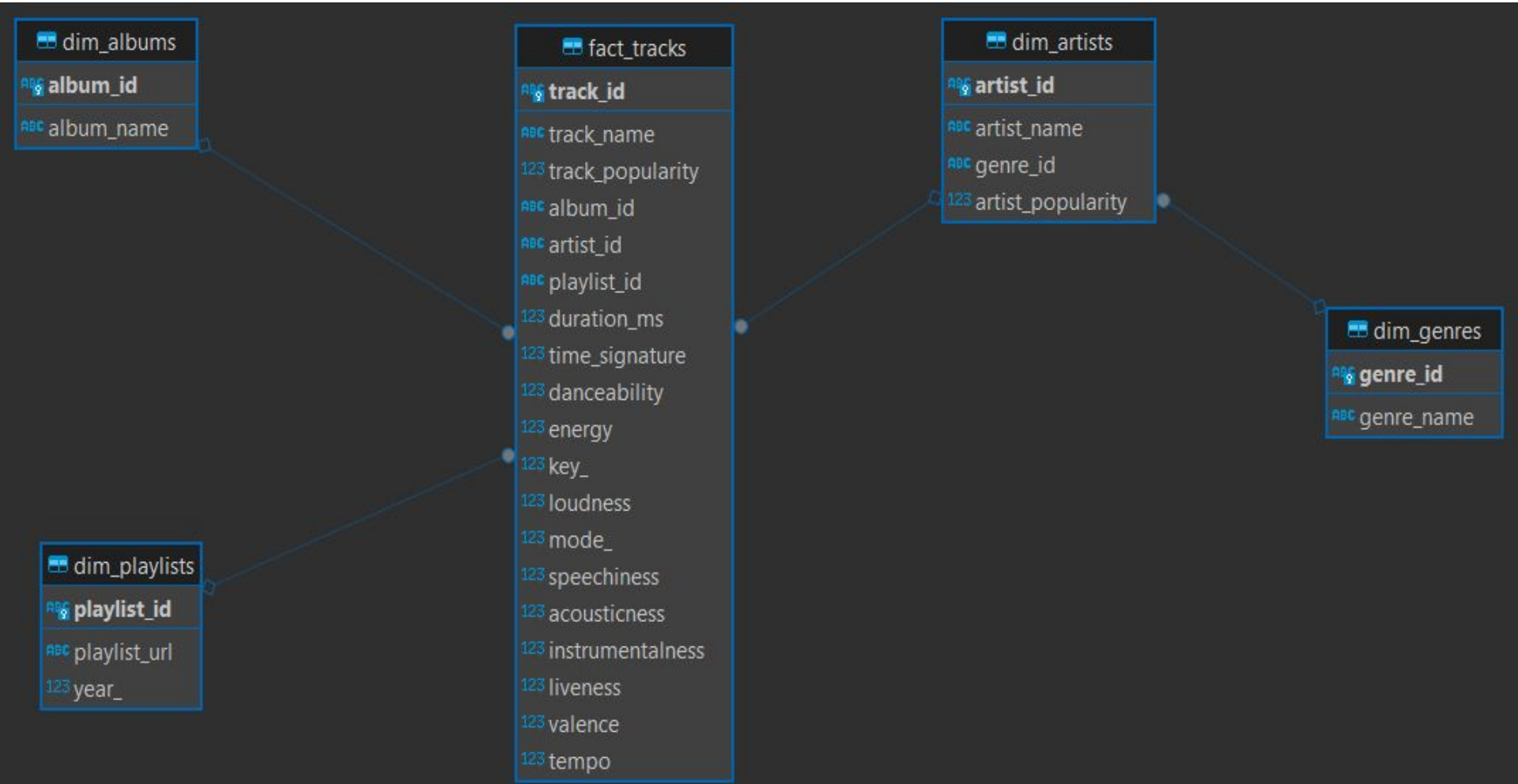
In the dataset, the main characteristics of the data are around the “tracks” table by referring to “artists”, “albums”, “genres”, and” playlists”, making it a perfect fact table in the denormalized schema.

Let’s explore the project!

The Entity Relation Diagram (ERD) for the raw Spotify data in 3NF



The Entity Relation Diagram (ERD) for the denormalized tables



Three different business analyses

1. For each music genre in our dataset, we can collect average track danceability metrics
2. We can display top three the most popular tracks for each playlist
3. For each playlist, we can query track popularity in cumulative distribution

Average track danceability for each genre metrics

```
-- Selecting average track danceability for each genre
-- With this query, we can collect all danceable genres and give personalized recommendations
select
	distinct genre_name,
	AVG(danceability) over (partition by g.genre_id) as avg_danceability
from
	denormalized_model.fact_tracks t
join denormalized_model.dim_artists art on
	art.artist_id = t.artist_id
join denormalized_model.dim_genres g on
	g.genre_id = art.genre_id
order by
	avg_danceability desc
limit 10
```

genre_name	avg_danceability
trap queen	0.937
r&b en espanol	0.919
melodic drill	0.911
deep underground hip hop	0.905
australian house	0.902
pop r&b	0.9
bedroom soul	0.878
urbano latino	0.87
australian pop	0.864
urban contemporary	0.856

Top three the most popular tracks for each playlist

```
-- Displaying top 3 the most popular tracks for each playlist
-- Using DENSE_RANK to avoid any unnecessary skips
select
	*
from
	((
	select
		track_id,
		pls.playlist_url,
		trc.track_popularity,
		dense_rank() over (partition by trc.playlist_id
	order by
		track_popularity desc ) as Row_Id
	from
		denormalized_model.fact_tracks trc
	join denormalized_model.dim_playlists pls on
		trc.playlist_id = pls.playlist_id)) as tp
where
	Row_Id <= 3 limit 10
```

track_id	playlist_url	track_popularity	row_id
0ct6r3EGTcMLPtrXHDvVjc	https://open.spotify.com/playlist/37i9dQZF1DX9ukdrXQLJGZ	88	1
6K4t31amVTZDgR3sKmwUJJ	https://open.spotify.com/playlist/37i9dQZF1DX9ukdrXQLJGZ	87	2
5E30LdtzQTGqRvNd7l6kG5	https://open.spotify.com/playlist/37i9dQZF1DX9ukdrXQLJGZ	86	3
7MXVkk9YMctZqd1Srtv4MB	https://open.spotify.com/playlist/37i9dQZF1DX8XZ6AUo9R4R	92	1
3xKsf9qdS1CyvXSMEid6g8	https://open.spotify.com/playlist/37i9dQZF1DX8XZ6AUo9R4R	90	2
1zi7xx7UVEFkmKfv06H8x0	https://open.spotify.com/playlist/37i9dQZF1DX8XZ6AUo9R4R	89	3
3JvKfv6T31zO0ini8iNItO	https://open.spotify.com/playlist/37i9dQZF1DX3Sp0P28SIer	91	1
2QjOHCTQ1Jl3zawyYOpxh6	https://open.spotify.com/playlist/37i9dQZF1DX3Sp0P28SIer	91	1
086myS9r57YsLbJpU0TgK9	https://open.spotify.com/playlist/37i9dQZF1DX3Sp0P28SIer	89	2
5FVd6KXrgO9B3JPmC8OPst	https://open.spotify.com/playlist/37i9dQZF1DX3Sp0P28SIer	89	2

Track popularity in CUME_DIST

```
-- Track popularity in cumulative distribution(CUME_DIST) grouped for each playlist

select
	trc.track_popularity,
	trc.playlist_id,
	ROUND((cume_dist() over (partition by trc.playlist_id
order by
	trc.track_popularity))::numeric,
	3) as cume_dist
from
	denormalized_model.fact_tracks trc limit 10
```

track_popularity	playlist_id	cume_dist
68	9wnrC2T34VLxtL7vdC8hVE	0.4
68	9wnrC2T34VLxtL7vdC8hVE	0.4
68	9wnrC2T34VLxtL7vdC8hVE	0.4
68	9wnrC2T34VLxtL7vdC8hVE	0.4
68	9wnrC2T34VLxtL7vdC8hVE	0.4
68	9wnrC2T34VLxtL7vdC8hVE	0.4
77	9wnrC2T34VLxtL7vdC8hVE	0.533
77	9wnrC2T34VLxtL7vdC8hVE	0.533
80	9wnrC2T34VLxtL7vdC8hVE	0.667
80	9wnrC2T34VLxtL7vdC8hVE	0.667

Problems

I found nothing problematic during the project but rather challenging and fun :)

Useful links:

- Github repo
- Dimensional modelling

Thank you for your time

Contacts

Linkedin Github Telegram