Practice on Database Normalization

Data Engineering

Here we have track info data (from Spotify data), which we got by normalizing the corresponding json data.

Table: top_track_info						
artist_name	**artist_id**	**song_id**	**song_name**	**streams**	**genre**	**followers**
Dance w/t Dead	x88928	a99189	diabolic	75092	[synthwave,darkwave]	156000
Dance w/t Dead	x88928	a73198	invader	93910	[synthwave,darkwave]	156000
Frank Sinatra	z99029	a23812	witchcraft	21048820	traditional	12020900
Frank Sinatra	z99029	a83012	angel eyes	14029901	traditional	12020900
ODESZA	y88420	a01818	Meridian	5401928	[electronic, indie]	6800700
ODESZA	y88420	a01912	Bloom	5691321	[electronic,indie]	6800700

The genre column is not **atomic** therefore top_track_info is not in 1st normal form.

In general, non-normal data

- Might be good for analysis
- Is a common format for CSV/excel
- Has lots and lots of extra info
- Difficult to update/change

Table: top_track_info							
artist_name	**artist_id**	**song_id**	**song_name**	**streams**	**genre_1**	**genre_2**	**followers**
Dance w/t Dead	x88928	a99189	diabolic	75092	synthwave	darkwave	156000
Dance w/t Dead	x88928	a73198	invader	93910	synthwave	darkwave	156000
Frank Sinatra	z99029	a23812	witchcraft	21048820	traditional		12020900
Frank Sinatra	z99029	a83012	angel eyes	14029901	traditional		12020900
ODESZA	y88420	a01818	Meridian	5401928	electronic	indie	6800700
ODESZA	y88420	a01912	Bloom	5691321	electronic	indie	6800700

We can add one column for each genre, however, we will increase the number of NULL values as each rows may not have 2 genre.

Also a row may have more than 2 gerne, so two columns may not be enough, so we have to add more columns based on the largest number genre associated with a song. This may exacerbate the problem of NULL values.

To correctly handle this attribute we have to look at functional dependencies in our data

Let’s look at the functional dependencies:

artist_id->artist_name,followers,genre (i.e. knowing the artist_id we can uniquely identify the artist name, their number of followers, and the genre of the artist)
song_id->song_name,streams (i.e. knowing the song_id, we can uniquely identify the song_name and number of streams)

song_id,artist_id-> all attributes (so song_id, artist_id is the key)

Table: top_track_info

artist_name	artist_id	song_id	song_name	streams	genre	followers
Dance w/t Dead	x88928	a99189	diabolic	75092	[synthwave,darkwave]	156000
Dance w/t Dead	x88928	a73198	invader	93910	[synthwave,darkwave]	156000
Frank Sinatra	z99029	a23812	witchcraft	21048820	traditional	12020900
Frank Sinatra	z99029	a83012	angel eyes	14029901	traditional	12020900
ODESZA	y88420	a01818	Meridian	5401928	[electronic, indie]	6800700
ODESZA	y88420	a01912	Bloom	5691321	[electronic,indie]	6800700

This relation is not in 1st normal form since genre is multi-value attribute

Since artist_id->genre, we can decompose the original relation based on this fd to track_info and genre_info:

Table: top_track_info						
artist_name	**artist_id**	**song_id**	**song_name**	**streams**	**genre**	**followers**

Table: track_info					
artist_name	**artist_id**	**song_id**	**song_name**	**streams**	**followers**

Table: genre_info	
artist_id	**genre**
x88928	synthwave
x88928	darkwave
z99029	traditional
y88420	electronic
y88420	indie

Finally we have flattened the genre values. After flattening the multi-value attribute we will have **artist_id,genre->artist_id,genre** therefore artist_id and genre is the key of the new genre_info relation and the new relation is in 3nf, so it does not need any more normalization.

In practice 3nf relations are good enough as a target form for normalization, so we try to create 3nf relations.

Now, lets look at track_info again. There is no multi-value attribute here so track_info is in 1NF.

Table: track_info					
artist_name	**artist_id**	**song_id**	**song_name**	**streams**	**followers**
Dance w/t Dead	x88928	a99189	diabolic	75092	156000
Dance w/t Dead	x88928	a73198	invader	93910	156000
Frank Sinatra	z99029	a23812	witchcraft	21048820	12020900
Frank Sinatra	z99029	a83012	angel eyes	14029901	12020900
ODESZA	y88420	a01818	Meridian	5401928	6800700
ODESZA	y88420	a01912	Bloom	5691321	6800700

We have **song_id->song_name,streams** and **artist_id->artist_name,followers**

We also can see that (song_id, artist_id) uniquely identify all rows, so artist_id, song_id -> track_info (i.e. all attributes)
Therefore artist_id is a prime attribute

We have:
artist_id->artist_name,followers
and
artist_id, song_id-> artist_name (based on the fact that artist_id, song_id is a key)

Based on definition of partial dependencies, artist_name is partially dependent on (artist_id, song_id) therefore track_info is not in 2nf

Now we have to decompose track_info:

Assume A,C → B and A → B and {A,C} is a candidate key
and B is not a prime attribute. Construct two R1 and R2 such that

R1: A^+

R2: $R - A^+ \cup A$

The closure of artist_id is artist_id, artist_name, and followers, therefore the new relations are

Table: artist_info

artist_name	artist_id	followers

Table: song_info

artist_id	song_id	song_name	streams

Let’s look at the new relations and their functional dependencies:

Table: artist_info		
artist_name	**artist_id**	**followers**
Dance w/t Dead	x88928	156000
Frank Sinatra	z99029	12020900
ODESZA	y88420	6800700

artist_id->artist_name, followers

Therefore artist_id is the key of artist_info relation

Since there is no partial dependencies any more, artist_info is in 2nf at least.

Since artist_info is in 2nf and there is no transitive functional dependency, artist_info is in **3nf**.

Table: song_info			
artist_id	**song_id**	**song_name**	**streams**
x88928	a99189	diabolic	75092
x88928	a73198	invader	93910
z99029	a23812	witchcraft	21048820
z99029	a83012	angel eyes	14029901
y88420	a01818	Meridian	5401928
y88420	a01912	Bloom	5691321

song_id->song_name, streams, artist_id

Since there is no partial dependency any more, song_info is at least in 2nf.

Since song_info is in 2nf and there is no transitive functional dependency, artist_info is in **3nf**.

The final set of relations:

Table: artist_info		
artist_name	**artist_id**	**followers**
Dance w/t Dead	x88928	156000
Frank Sinatra	z99029	12020900
ODESZA	y88420	6800700

Table: genre_info	
artist_id	**genre**
x88928	synthwave
x88928	darkwave
z99029	traditional
y88420	electronic
y88420	indie

Table: song_info			
artist_id	**song_id**	**song_name**	**streams**
x88928	a99189	diabolic	75092
x88928	a73198	invader	93910
z99029	a23812	witchcraft	21048820
z99029	a83012	angel eyes	14029901
y88420	a01818	Meridian	5401928
y88420	a01912	Bloom	5691321

Database Schema:

artist_id is key of artist_info relation therefore it becomes foreign key at genre_info and song_info

This diagram that shows the relations and connections of foreign key-primary key pairs is called database schema, which is all you need to create the tables.

Table: artist_info

artist_name	artist_id	followers
Dance w/t Dead	x88928	156000
Frank Sinatra	z99029	12020900
ODESZA	y88420	6800700

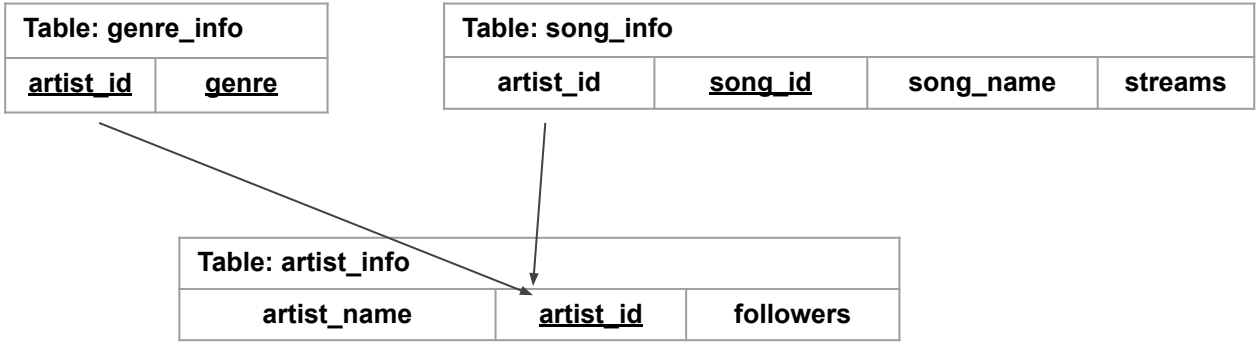
Table: genre_info

artist_id	genre
x88928	synthwave
x88928	darkwave
z99029	traditional
y88420	electronic
y88420	indie

Table: song_info

artist_id	song_id	song_name	streams
x88928	a99189	diabolic	75092
x88928	a73198	invader	93910
z99029	a23812	witchcraft	21048820
z99029	a83012	angel eyes	14029901
y88420	a01818	Meridian	5401928
y88420	a01912	Bloom	5691321

Final schema:



In the schema, the primary key attributes are shown using underline, and arrows indicate foreign key-primary key relations