Let's take a moment to make sure you understand what was in the demo regarding denormalized vs. normalized data. These are important concepts, so make sure to spend some time reflecting on these.

Normalization is about trying to increase data integrity by reducing the number of copies of the data. Data that needs to be added or updated will be done in as few places as possible.

Denormalization is trying to increase performance by reducing the number of joins between tables (as joins can be slow). Data integrity will take a bit of a potential hit, as there will be more copies of the data (to reduce JOINS).

Example of Denormalized Data:

As you saw in the earlier demo, this denormalized table contains a column with the Artist name that includes duplicated rows, and another column with a list of songs.

Album ID	Album Name	Artist Name	Year	List of Songs
1	Rubber Soul	The Beatles	1965	['Michelle', 'Think For Yourself', 'In My Life']
2	Let it Be	The Beatles	1970	['Let it Be' , 'Across The Universe']

Example of Normalized Data:

Now for normalized data, Amanda used 3NF. You see a few changes:
1) *No row contains a list of items*. For e.g., the list of song has been replaced with each song having its own row in the Song table.
2) *Transitive dependencies have been removed*. For e.g., album ID is the PRIMARY KEY for the album year in Album Table. Similarly, each of the other tables have a unique primary key that can identify the other values in the table (e.g., song id and song name within Song table).

Song_Table

Song ID	Album ID	Song Name
1	1	Michelle
2	1	Think For Yourself
3	1	In My Life
4	2	Let it Be
5	2	Across The Universe

Album_Table

Album ID	Album Name	Artist ID	Year
1	Rubber Soul	1	1965
2	Let it Be	1	1970

Artist_Table

Artist ID	Artist Name
1	The Beatles