

# MUSIC APP DATABASE MANAGEMENT SYSTEM

- Ashish Kumar Singh (2019BCS-010)
- Darsh Kevin Shah (2019BCS-015)
- Tekumudi Jnana Lakshmi (2019BCS-066)

# PROBLEM STATEMENT

A music app is planning to implement a database to enhance its data management practice and ultimately advance its business operations.

The initial planning analysis phases have revealed the following system requirements:

- Each album has a unique Album ID as well as the following attributes: Album Title, Year. An album contains at least one song or more songs.
- Songs are identified by Song ID.
- Each song can be contained in more than one album or not contained in any of them at all and has a Song Title, Year and Duration.
- Each song belongs to at least one genre or multiple genres.

- Songs are written by at least an artist or multiple artists.
- Each artist has a unique Artist ID corresponding to his or her name, and an artist writes at least one song or multiple songs, to be recorded in the database or releases one or multiple Albums on his own or in collaboration with different artists.
- Also many artists can collaborate to write a song together which may or may not belong to their solo albums.
- To use the app, one needs to first sign up on the app as a user giving details of his name, date of birth, phone number.
- Each user is identified by a unique User ID.

- Also the user can subscribe to a premium version of the app to get rid off the advertisements on the app and enjoy features like offline playback.
- To subscribe to a premium version of the app the user has to select a plan. Each Premium Plan has its own unique ID corresponding to the duration of the plan and price.
- The details such as method of payment and date of subscription are also maintained.
- Also the user account has other attributes like Liked Songs, Liked Albums, Playlists and Frequents.

# ENTITY RELATIONSHIP DIAGRAM

The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

It consists of Entities, their Attributes and the Relationship between the Entities.

**Entity** : An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. In our project we took Album, Song, Genre, Playlist, Premium Plan, User and Artist as entities. They are represented by rectangular blocks.

**Attributes** : Entities are represented by means of their properties, called **attributes**. All attributes have values.

There are different types of attributes:

- Simple attribute
- Composite attribute
- Derived attribute
- Single-value attribute
- Multi-value attribute

Some of the attributes in the ER diagram are Song ID, Duration, Date of Birth, Duration of Plan, Frequency, Album Title, etc. They are represented by ellipses.

**Keys** : Key is an attribute or collection of attributes that uniquely identifies an entity among entity set. We have different types of Keys which are:

- Super Key
- Candidate Key
- Primary Key

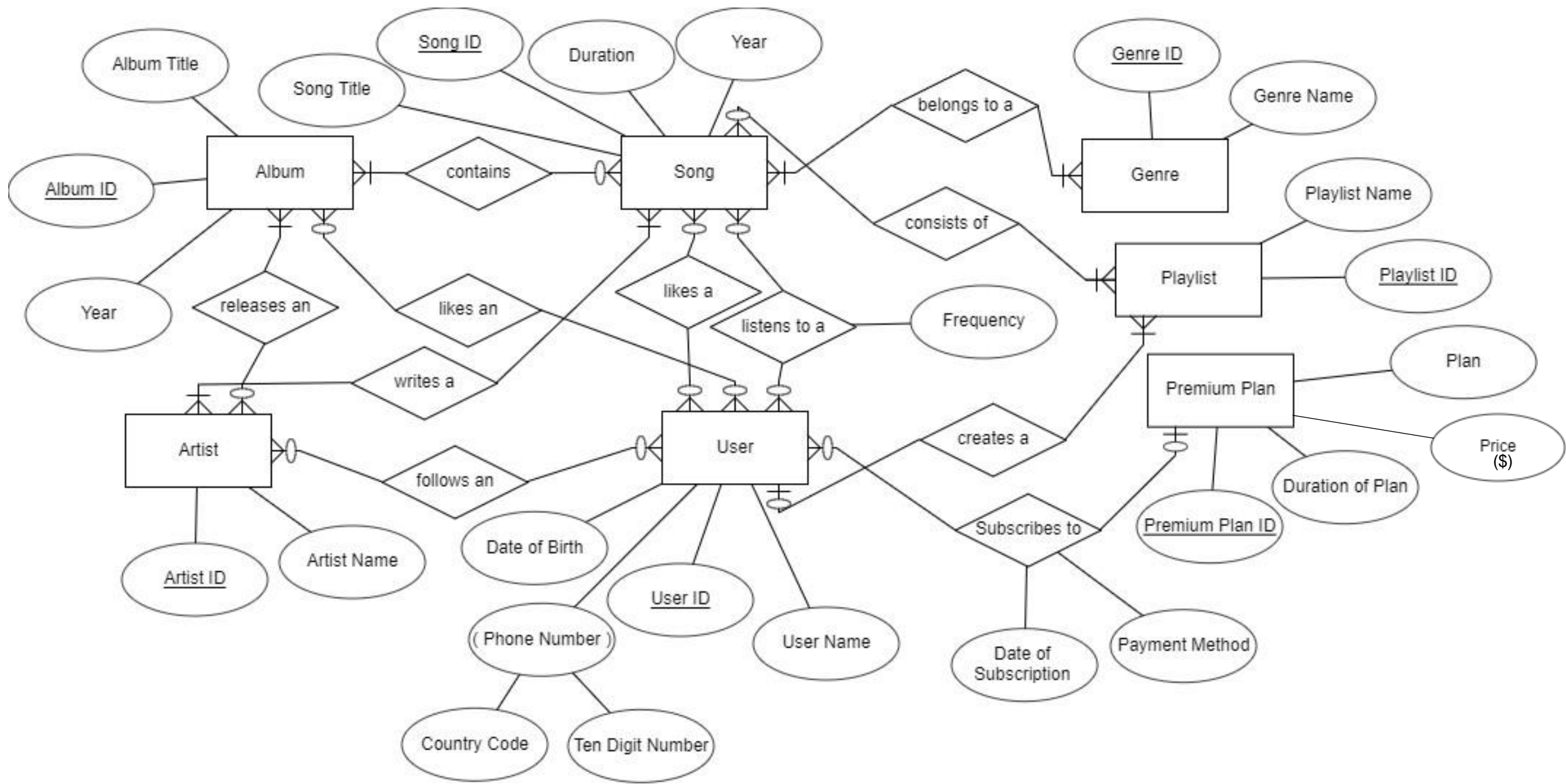
In the ER model, the attributes that are Keys are underlined.

**Relationship** : The association among entities is called a relationship.

Relationships can also be categorized based on it's degree as:

- Binary = degree 2
- Ternary = degree 3
- n-ary = degree





# CONVERSION OF ER DIAGRAM TO RELATIONAL SCHEMA

ER Model, when conceptualized into diagrams, gives a good overview of entity-relationship, which is easier to understand. ER diagrams can be mapped to relational schema, that is, it is possible to create relational schema using ER diagram. We cannot import all the ER constraints into relational model, but an approximate schema can be generated.

We can easily achieve this by the Mapping process.

### 1. Mapping Entities:

The algorithm behind this is

- Create table for each entity.
- Entity's attributes should become fields of tables with their respective data types.
- Declare primary key.

## 2. Mapping Relationship

It can be done by:

- Create table for a relationship.
- Add the primary keys of all participating Entities as fields of table with their respective data types.
- If relationship has any attribute, add each attribute as field of table.
- Declare a primary key composing all the primary keys of participating entities.
- Declare all foreign key constraints.

## 3. Mapping Weak Entity Sets

The steps are:

- Create table for weak entity set.
- Add all its attributes to table as field.
- Add the primary key of identifying entity set.

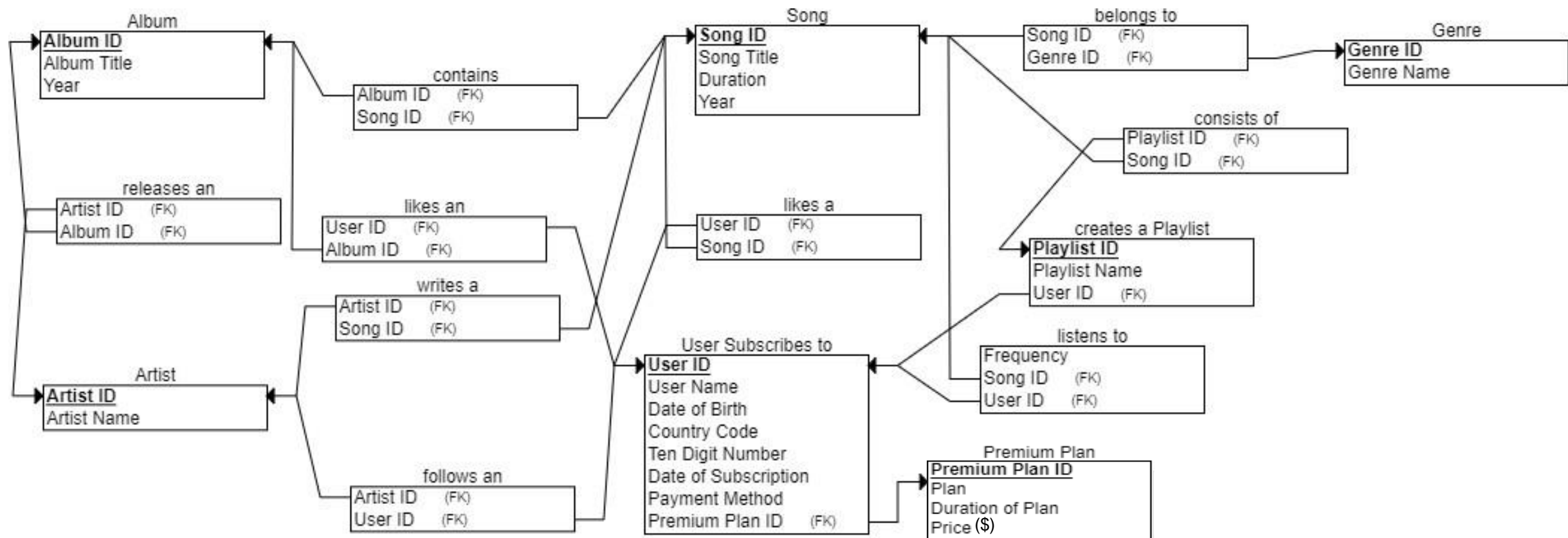
- Declare all foreign key constraints.

#### 4. Mapping Hierarchical Entities

This includes:

- Create tables for all higher-level entities.
- Create tables for lower-level entities.
- Add primary keys of higher-level entities in the table of lower-level entities.
- In lower-level tables, add all other attributes of lower-level entities.
- Declare primary key of higher-level table and the primary key for lower-level table.
- Declare foreign key constraints.

After all these steps we get the relational schema.



# NORMALIZED TABLES

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

- **Update anomalies** – If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.
- **Deletion anomalies** – We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
- **Insert anomalies** – We tried to insert data in a record that does not exist at all.

Normalization is a method to remove all these anomalies and bring the database to a consistent state.



## First Normal Form:

First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains.

The values in an atomic domain are indivisible units.

Each attribute must contain only a single value from its pre-defined domain.

For example: We have two values in SongName , 'Normal' and 'Lucky you'. So you would make a new tuple(row) dividing the previous one.

So it would be (SongName , 'Normal') and (SongName , 'Lucky you')

## Second Normal Form:

It uses the concept of **Prime attribute** and **Non-prime attribute**.

If we follow second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if  $X \rightarrow A$  holds, then there should not be any proper subset  $Y$  of  $X$ , for which  $Y \rightarrow A$  also holds true.

### Third Normal Form:

For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy –

- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency,  $X \rightarrow A$ , then either –
  - $X$  is a superkey or,
  - $A$  is prime attribute.

### Boyce-Codd Normal Form:

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that –

- For any non-trivial functional dependency,  $X \rightarrow A$ ,  $X$  must be a super-key.

**Belongs to**

Song ID	Genre ID
1	1
1	2
2	1
2	2
3	1
3	2
4	3
4	5
5	3
6	3
7	1
7	2
8	1
8	2
9	1
9	2
10	5
11	5
12	4
13	6
14	3
15	1
15	2
15	4
16	3
17	5

**Song**

Song ID	Song Title	Duration	Year
1	The Ringer	00:05:38	2018
2	Normal	00:03:43	2018
3	Lucky You	00:04:05	2018
4	Some Nights	00:04:37	2012
5	We Are Young	00:04:10	2012
6	Why Am I the One	00:04:46	2012
7	Rap God	00:06:03	2013
8	The Monster	00:04:10	2013
9	Headlights	00:05:43	2013
10	American Idiot	00:02:54	2004
11	Wake me up when September Ends	00:07:13	2004
12	Holiday/Boulevard of Broken Dreams	00:08:13	2004
13	Alone	00:03:20	2016
14	Viva La Vida	00:05:19	2008
15	Heathens	00:03:15	2016
16	Complicated	00:03:04	2017
17	bad guy	00:03:14	2019

**Genre**

Genre ID	Genre Name
1	Hip-Hop
2	Rap
3	Pop
4	Rock
5	Indie
6	Dance/Electronic

Artist

Artist ID	Artist Name
1	Eminem
2	Joyner Lucas
3	fun.
4	Janelle Monáe
5	Rihanna
6	Nate Ruess
7	Green Day
8	Marshmello
9	Coldplay
10	Twenty One Pilots
11	David Guetta
12	Dimitri Vegas
13	Like Mike
14	Billie Eilish

Premium Plan

Premium Plan ID	Plan	Duration of Plan	Price(\$)
1	Value Pack	1	5
2	Value Pack	3	10
3	Value Pack	6	15
4	Value Pack	9	20
5	Value Pack	12	25

Album

Album ID	Album Title	Year
1	Kamikaze	2018
2	Some Nights	2012
3	The Marshall Mathers LP2	2013
4	American Idiot	2004

Contains

Album ID	Song ID
1	1
1	2
1	3
2	4
2	5
2	6
3	7
3	8
3	9
4	10
4	11
4	12

Releases an

Artist ID	Album ID
1	1
3	2
1	3
7	4

## Writes a

Artist ID	Song ID
1	1
1	2
1	3
2	3
3	4
3	5
4	5
3	6
1	7
1	8
5	8
1	9
6	9
7	10
7	11
7	12
8	13
9	14
10	15
11	16
12	16
13	16
14	17

## User Subscribes to

User ID	User Name	Date of Birth	Country Code	Ten Digit Number	Date of Subscription	Payment Method	Premium Plan ID
1	Aryan Chauhan	2001-08-04	+91	8836885789	2020-05-13	UPI	1
2	Dhruv Datta	2001-10-30	+91	9818896547	null	null	null
3	Aryan Anand	2001-06-24	+44	8368975098	null	null	null
4	Arjun Sharma	2001-09-05	+91	9868723161	2020-05-15	Paytm	3

## Follows an

User ID	Artist ID
1	1
1	9
1	11
2	9
2	11
4	1
4	3
4	9
4	11
4	8

## Likes an

User ID	Album ID
1	1
1	3
2	2
2	4
3	4
4	1
4	2

## Likes a

User ID	Song ID
1	7
1	8
1	12
1	13
2	2
2	9
3	14
3	16
4	4
4	5
4	7

## Listens to

User ID	Song ID	Frequency
1	4	7
1	7	14
1	8	29
1	9	1
1	10	24
1	11	28
2	1	3
2	2	16
2	3	25
2	7	19
2	8	25
2	9	12
3	4	30
3	5	34
3	6	37
3	14	20
3	17	10
4	7	11
4	8	16
4	9	2
4	13	21
4	15	33

## Creates a Playlist

Playlist ID	Playlist Name	User ID
1	BANGERS	1
2	Rap	1
3	Rock Classics	2
4	EDM	3
5	2010s	4

## Consists of

Playlist ID	Song ID
1	8
1	9
1	14
1	15
1	16
2	1
2	2
2	3
2	7
3	10
3	11
3	12
4	13
4	16
5	4
5	5
5	6
5	14

# FUNCTIONAL DEPENDENCIES

**Functional dependency** (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes  $A_1, A_2, \dots, A_n$ , then those two tuples must have to have same values for attributes  $B_1, B_2, \dots, B_n$ .

Functional dependency is represented by an arrow sign ( $\rightarrow$ ) that is,  $X \rightarrow Y$ , where  $X$  functionally determines  $Y$ . The left-hand side attributes determine the values of attributes on the right-hand side.

### **Armstrong's Axioms:**

If  $F$  is a set of functional dependencies then the closure of  $F$ , denoted as  $F^+$ , is the set of all functional dependencies logically implied by  $F$ . Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

- **Reflexive rule** – If  $\alpha$  is a set of attributes and  $\beta$  is\_subset\_of  $\alpha$ , then  $\alpha$  holds  $\beta$ .



- **Augmentation rule** – If  $a \rightarrow b$  holds and  $y$  is attribute set, then  $ay \rightarrow by$  also holds. That is adding attributes in dependencies, does not change the basic dependencies.
- **Transitivity rule** – Same as transitive rule in algebra, if  $a \rightarrow b$  holds and  $b \rightarrow c$  holds, then  $a \rightarrow c$  also holds.  $a \rightarrow b$  is called as a functionally that determines  $b$ .

Types of functional dependencies:

- Multivalued dependency:
- Trivial functional dependency:
- Non-trivial functional dependency:
- Transitive dependency:

- **Table name : belongs to**

Only has trivial functional dependencies, i.e.,

1. Song ID  $\rightarrow$  Song ID
2. Genre ID  $\rightarrow$  Genre ID

- **Table name : Song**

Has trivial as well as non trivial functional dependencies. The non trivial dependencies are:

1. Song ID  $\rightarrow$  Song Title
2. Song ID  $\rightarrow$  Duration
3. Song ID  $\rightarrow$  Year
4. Song Title  $\rightarrow$  Duration
5. Duration  $\rightarrow$  Year
6. (Song ID, Song Title)  $\rightarrow$  Duration
7. (Song ID, Duration)  $\rightarrow$  Song Title

and so on. Notice that Year and another column cannot be a functional dependency.

- **Table Name : Genre**

The Functional dependencies are:

1. Genre ID  $\rightarrow$  Genre Name
2. Genre Name  $\rightarrow$  Genre ID

and the various combinations.

- **Table Name : Artist**

Has the following functional dependencies:

1. Artist ID  $\rightarrow$  Artist Name
2. (Artist ID, Artist Name)  $\rightarrow$  Artist Name

and the various combinations, as it is a small table.

- **Table Name : Premium Plan**

The functional dependencies are:

1. Premium Plan ID  $\rightarrow$  Plan
2. (Plan, Duration of the plan)  $\rightarrow$  Price
3. (Premium Plan ID, Plan)  $\rightarrow$  (Duration of the plan, Plan, Price)
4. Price  $\rightarrow$  Plan

and so on..

- **Table Name : Contains, Album**

These two tables contain the following:

1. Album ID → Album Title
2. (Album ID, Album Title) → Year
3. Year → Album ID
4. Year → (Album Title, Album ID)
5. Song ID → Album ID (in Contains)

- **Table Name : User Subscribers to**

In this table Date of Subscription, Payment method and Premium Plan ID cannot have functional dependencies as they have null values.

1. (User ID, Date of Birth, Country Code) → Ten digit Number
  2. Date of Birth → (Country Code, Ten Digit Number)
  3. User Name → Country Code
- and so on.. Are some of the functional dependencies

- **Table Name : Follows an, Likes an, Consists of**

In these two tables, if you notice, there are only trivial functional dependencies.

- **Table Name : Listens to**

Some of the functional dependencies are:

1. (User ID, Song ID)  $\rightarrow$  Frequency
2. (User ID, Frequency)  $\rightarrow$  Song ID
3. Frequency  $\rightarrow$  User ID

- **Table Name : Creates a Playlist**

Has the following functional dependencies:

1. Playlist ID  $\rightarrow$  (Playlist Name, User ID)
2. (Playlist Name, User ID)  $\rightarrow$  (Playlist ID, User ID)
3. Playlist Name  $\rightarrow$  (Playlist Name, Playlist ID, User ID)
4. Playlist ID  $\rightarrow$  Playlist Name
5. Playlist ID  $\rightarrow$  User ID
6. User ID  $\rightarrow$  User ID (trivial functional dependency)

# SOME BASIC QUERIES

## Creating the database:

```
1 • CREATE DATABASE MusicApp;  
2 • USE MusicApp;
```

## Creating a Table called Song:

Along with creating a table, we add a few attributes like SongID, SongTitle, Duration time, SongYear and their constraints making keys too.

In the next lines we **insert** the tuples with the respective entries.

```
4 • CREATE TABLE Song(SongID int NOT NULL PRIMARY KEY, SongTitle text NOT NULL,  
5   Duration time NOT NULL, SongYear int NOT NULL);  
6 • INSERT INTO Song VALUES(1, 'The Ringer', '00:05:38', 2018);  
7 • INSERT INTO Song VALUES(2, 'Normal', '00:03:43', 2018);  
8 • INSERT INTO Song VALUES(3, 'Lucky You', '00:04:05', 2018);  
9 • INSERT INTO Song VALUES(4, 'Some Nights', '00:04:37', 2012);  
10 • INSERT INTO Song VALUES(5, 'We Are Young', '00:04:10', 2012);  
11 • INSERT INTO Song VALUES(6, 'Why Am I the One', '00:04:46', 2012);  
12 • INSERT INTO Song VALUES(7, 'Rap God', '00:06:03', 2013);  
13 • INSERT INTO Song VALUES(8, 'The Monster', '00:04:10', 2013);
```

## Creating more Tables:

We further build up the database by adding a few more tables.







```
100 • INSERT INTO releases VALUES(1,1);  
101 • CREATE TABLE writesA(ArtistID int NOT NULL, SongID int NOT NULL, FOREIGN KEY(ArtistID)  
102 REFERENCES Artist(ArtistID), FOREIGN KEY(SongID) REFERENCES Song(SongID),  
103 PRIMARY KEY(ArtistID, SongID));  
104 • INSERT INTO writesA VALUES(1,1);  
105 • INSERT INTO writesA VALUES(1,2);  
106 • INSERT INTO writesA VALUES(1,3);  
107 • INSERT INTO writesA VALUES(2,3);  
108 • INSERT INTO writesA VALUES(3,4);  
109 • INSERT INTO writesA VALUES(3,5);  
110 • INSERT INTO writesA VALUES(4,5);  
111 • INSERT INTO writesA VALUES(3,6);
```

## To display a Table:

```
207  
208 • SELECT * FROM Song;  
209
```






We get the following output

Result Grid				
Filter Rows: <input type="text"/>				
Edit:   				
Export/Import:  				
Wrap Cell Content: 				
	SongID	SongTitle	Duration	SongYear
▶	1	The Ringer	00:05:38	2018
	2	Normal	00:03:43	2018
	3	Lucky You	00:04:05	2018
	4	Some Nights	00:04:37	2012
	5	We Are Young	00:04:10	2012
	6	Why Am I the One	00:04:46	2012
	7	Rap God	00:06:03	2013
	8	The Monster	00:04:10	2013
	9	Headlights	00:05:43	2013

Using conditions:

```
210 • SELECT * FROM Song
211 WHERE SongID > 6 AND SongID <= 14;
212
```

And we get the output as:

Result Grid				
Filter Rows: <input type="text"/>				
Edit:   				
	SongID	SongTitle	Duration	SongYear
▶	7	Rap God	00:06:03	2013
	8	The Monster	00:04:10	2013
	9	Headlights	00:05:43	2013
	10	American Idiot	00:02:54	2004
	11	Wake me up when September Ends	00:07:13	2004
	12	Holiday/Boulevard of Broken Dreams	00:08:13	2004
	13	Alone	00:03:20	2016
	14	Viva La Vida	00:05:19	2008
*	NULL	NULL	NULL	NULL




## Updating a Table:

```




213 • UPDATE Song
214     SET SongTitle='Lie To Me', SongYear='2018', Duration='2:30'
215     WHERE SongID=7;

```

Which changes the data from

Result Grid				
Filter Rows: <input type="text"/>				
Edit:   				
	SongID	SongTitle	Duration	SongYear
▶	7	Rap God	00:06:03	2013

to

Result Grid				
Filter Rows: <input type="text"/>				
Edit:   				
	SongID	SongTitle	Duration	SongYear
▶	7	Lie To Me	02:30:00	2018
*	NULL	NULL	NULL	NULL

## Deletion:

```
222 • DELETE FROM Song WHERE SongID = '7';
```

This would return an output stating that 1 row has been affected.

## To get an ordered output, on a condition:

This gives an ordered table bases on the sorted 'SongTitle' column.

```
224 • SELECT * FROM Song
225 ORDER BY SongTitle;
```

Part of the output is:



	SongID	SongTitle	Duration	SongYear
▶	13	Alone	00:03:20	2016
	10	American Idiot	00:02:54	2004
	17	bad guy	00:03:14	2019
	16	Complicated	00:03:04	2017
	9	Headlights	00:05:43	2013
	15	Heathens	00:03:15	2016
	12	Holidav/Boulevard of Broken Dreams	00:08:13	2004

To display a limited number of tuples from the top:

The code for this varies depending on the server. For MySQL the syntax is:

```
227 • SELECT SongTitle
228     FROM Song
229     WHERE songyear < 2017
230     LIMIT 7;
```

This shows us only the top 7 rows having the SongYear value less than 2017, or the songs that came before the year 2017




Result Grid			 Filter Rows:
	SongTitle		
▶	Some Nights		
	We Are Young		
	Why Am I the One		
	The Monster		
	Headlights		
	American Idiot		
	Wake me up when September Ends		

To get the total number of the occurrences of a specific value in a table:

We can use the `count()` function for this.

```
234 • SELECT COUNT(SongYear)
235 FROM Song
236 WHERE SongYear='2012';
```



And we get the output as 3, because there are three songs that were released in 2012 in the Table Song

Result Grid			 Filter
	COUNT(SongYear)		
	3		

For the minimum or maximum value:



The syntax and the output are:

```
238 • SELECT MIN(Frequency)
239     FROM ListensTo;
```

Result Grid				Filter Rows:
	MIN(Frequency)			
▶	1			

For minimum

```
241 • SELECT MAX(Frequency)
242     FROM ListensTo;
```

Result Grid				Filter Rows:
	MAX(Frequency)			
▶	37			

For maximum

There are many more such simple queries, and using these more complex queries can be executed.

# CONCLUSION

We first started by defining our problem. Almost every person listens to music and over the centuries there have been many songs and albums. So this clearly tells us that there must be a lot of data and we would eventually need a database to store and manage all of it.

Then we listed out all the entities and their attributes, formed relations between them and made the ER diagram. It was a little hard to make it at first but we could make it after referring to a few websites and video lectures.

Then we proceeded to convert the ER diagram to the relational schema. This was fairly easy.

To normalize the tables and get the functional dependencies we followed the manual method. Doing so we found that the database became more accurate and reduced the storage space.

Throughout the project we referred to various websites and the resources our professor Dr. Neetesh Kumar shared with us. We could understand the subject better doing this project hands-on and could learn how to apply SQL as well.



**THANK YOU**