ASSIGNMENT

ENTITY / RELATIONSHIP DIAGRAM

QUESTION:

Let's Design the Database of Spotify.

As a first step let's start with all the tables that are required to design the database of Spotify.

Here's a brief explanation of the use of each table in the database design for a system like Spotify:

- **Users table:** This table will store information about the registered users of the system, such as their name, email, password, date of birth, and profile image.
- **Artists table:** This table will store information about the artists, such as their names, genre, and image.
- **Albums table**: This table will store information about the albums, such as the album name, release date, and image. It will also have a foreign key linking to the Artists table.
- **Tracks table:** This table will store information about the tracks, such as the track name, duration, and file path. It will also have a foreign key linking to the album table.
- **Playlists table**: This table will store information about the playlists created by the users, such as the playlist name and image. It will also have a foreign key linking to the Users table.
- **Playlist_Tracks table:** This table will store the association between playlists and tracks. It will have two foreign keys, linking to the Playlists and Tracks tables.
- **Followers table:** This table will store the association between users and artists. It will have two foreign keys, linking to the Users and Artists tables.
- **Likes table**: This table will store information about the tracks liked by the users, such as the date and time of the like. It will have two foreign keys, linking to the Users and Tracks tables.

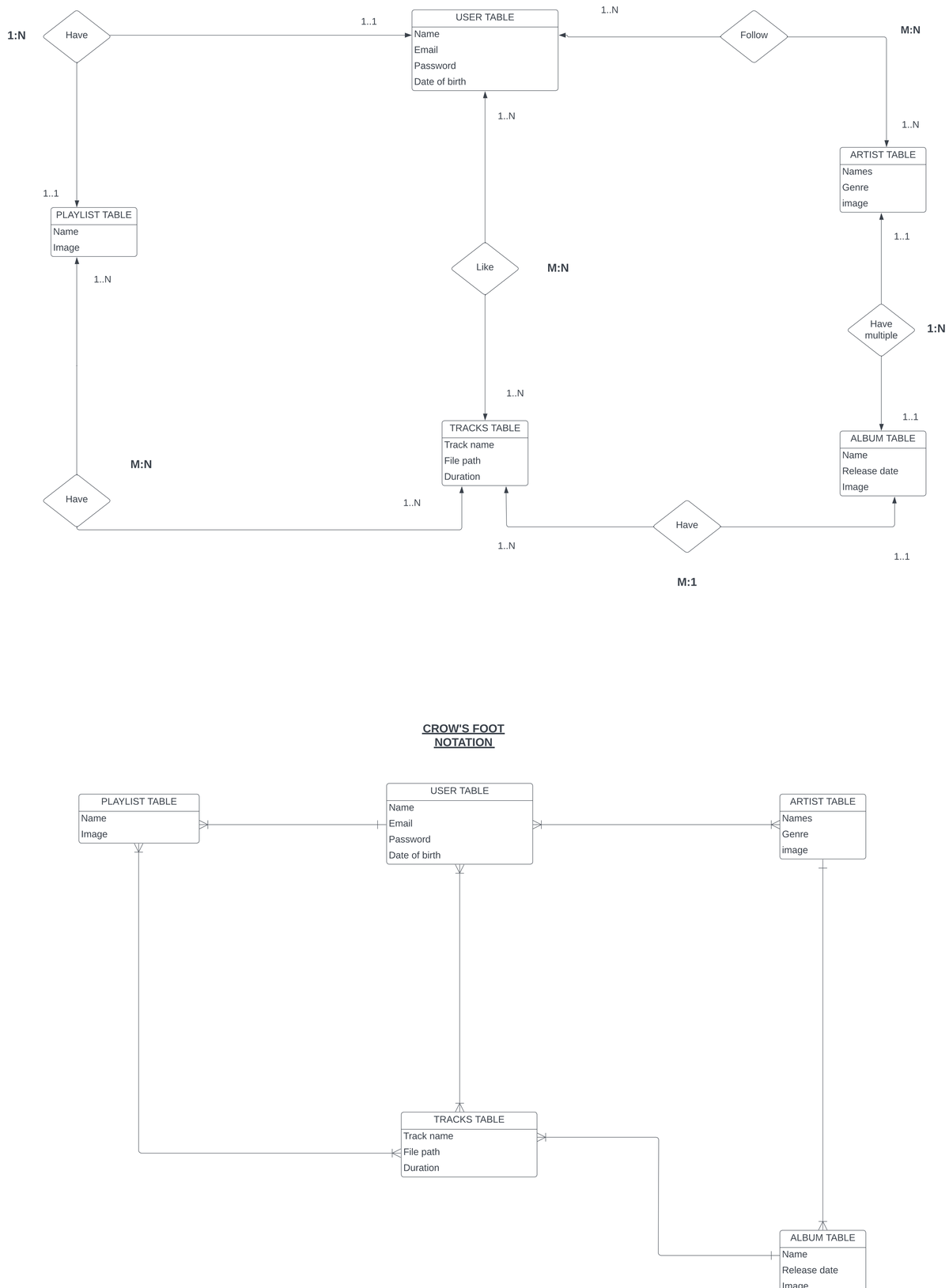
By using these tables, the system can easily retrieve and manipulate the data related to users, artists, albums, tracks, playlists, followers, and likes.

In the next step, Let's create the relationships between these tables.

Here is an example of how the relationships between the tables in the Spotify database design might be structured:

* One-to-Many relationship between Users and Playlists: One user can have multiple playlists.
* One-to-Many relationship between Artists and Albums: One artist can have multiple albums.
* One-to-Many relationship between Albums and Tracks: One album can have multiple tracks.
* Many-to-Many relationship between Playlists and Tracks: One playlist can have multiple tracks, and one track can be part of multiple playlists. The relationship is managed by the Playlist_Tracks table.
* Many-to-Many relationship between Users and Artists: One user can follow multiple artists, and one artist can be followed by multiple users. The relationship is managed by the Followers table.
* Many-to-Many relationship between Users and Tracks: One user can like multiple tracks, and one track can be liked by multiple users. The relationship is managed by the Likes table.

By using these relationships, the system can easily retrieve and manipulate the data related to users, artists, albums, tracks, playlists, followers, and likes.



RELATIONAL MODEL

- User Table (ID, Name, Email, password, DOB, and profile image)
- Artists Table (ID , Name,genre, artist image)
- Album Table (ID, Name, Release date, Image,# Artists Names)
- Tracks Table (ID, Track name, Duration, and FilePath, # Album Name)
- Playlists Table (ID,Name, Image, #Users Name)
- Playlists_Tracks Table(ID,Name, Genre, #Tracks Name , # Playlists Name)
- Followers Table (ID,Name, Gender, # Users Name, Artists Name)
- Likes Table (ID,Date, Time, #Users Name, # Tracks Name)

CROW'S FOOT NOTATION