USERCASE:

A use case for a system like Spotify would outline the interactions between users and the system, describing various scenarios and functionalities. Here are some example use cases for a music streaming platform:

1. **User Registration and Account Management**:
   * User creates an account.
   * User logs in and out.
   * User resets the password.
   * User updates account information (e.g., profile picture, email address).
2. **Music Playback**:
   * User searches for a specific song, artist, or album.
   * User plays a song, album, or playlist.
   * User skips to the next song.
   * User creates a custom playlist.
   * User shuffles a playlist.
   * User repeats a song.
3. **Music Discovery**:
   * User receives personalized song recommendations.
   * User explores new music by genre.
   * User views artist profiles and related artists.
4. **Playlist Management**:
   * User creates a new playlist.
   * User adds songs to a playlist.
   * User removes songs from a playlist.
   * User reorders songs within a playlist.
   * User shares a playlist with friends.
5. **Social Interaction**:
   * User follows other users.
   * User shares songs or playlists on social media.
   * User listens to playlists created by friends.
6. **Offline Listening (if supported)**:
   * User downloads songs for offline listening.
   * User manages downloaded content.
7. **User Engagement**:
   * User likes or dislikes a song.
   * User provides feedback on song recommendations.
   * User interacts with ads or premium features (if applicable).
8. **Settings and Preferences**:
   * User adjusts audio quality settings.
   * User configures notification preferences.
   * User sets streaming and download preferences (e.g., using Wi-Fi only).
9. **Support and Help**:
   * User accesses help resources.
   * User submits a support request.

Each use case would typically include a description of the scenario, the steps involved, any data exchanged, and any possible variations or alternative paths. Use cases serve as a foundation for system design, development, and testing, ensuring that the platform meets the needs and expectations of its users.

E/R diagram:

Designing an Entity-Relationship (E/R) diagram for a Spotify-like database involves defining the key entities and their relationships. Here's a simplified example of how the E/R diagram might look:

Entities:

1. **User**:
   * Attributes: UserID (Primary Key), Username, Email, Password, Date of Birth, Subscription Type, Profile Picture, etc.
2. **Artist**:
   * Attributes: ArtistID (Primary Key), Name, Genre, Nationality, etc.
3. **Album**:
   * Attributes: AlbumID (Primary Key), Title, Release Date, ArtistID (Foreign Key), Genre, Cover Art, etc.
4. **Song**:
   * Attributes: SongID (Primary Key), Title, Duration, ArtistID (Foreign Key), AlbumID (Foreign Key), Genre, Release Date, Audio File, etc.
5. **Playlist**:
   * Attributes: PlaylistID (Primary Key), Title, Description, UserID (Foreign Key), Created Date, Cover Image, etc.
6. **PlaylistSong** (Association Entity):
   * Attributes: PlaylistID (Foreign Key), SongID (Foreign Key), Order (to maintain song order within a playlist), etc.
7. **UserSongInteraction** (for tracking user listening history):
   * Attributes: InteractionID (Primary Key), UserID (Foreign Key), SongID (Foreign Key), Timestamp, Action (e.g., play, like, skip, share, etc.), etc.

Relationships:

* Users can create and own playlists, so there's a one-to-many relationship between the User entity and the Playlist entity.
* Users can interact with songs, resulting in a many-to-many relationship between the User entity and the Song entity, facilitated through the UserSongInteraction entity.
* Artists can create many albums, establishing a one-to-many relationship between the Artist entity and the Album entity.
* Artists can also create many songs, creating a one-to-many relationship between the Artist entity and the Song entity.
* Albums can contain multiple songs, indicating a one-to-many relationship between the Album entity and the Song entity.
* Playlists can include multiple songs, forming a many-to-many relationship between the Playlist entity and the Song entity, managed through the PlaylistSong association entity.

This is a simplified representation of a Spotify-like database, and a real-world system would likely have additional complexity and features. The E/R diagram provides a foundation for designing the database schema, but it's crucial to consider indexing, constraints, and optimization for performance in a production environment.

DATASET:

To create a music database similar to Spotify, you would need a variety of music-related datasets. Here are some recommended datasets to consider:

1. **Music Metadata**:
   * A dataset containing information about songs, including title, artist, album, genre, release date, and track duration.
2. **Artist Information**:
   * Data about artists, including their names, biographies, nationalities, and other relevant details.
3. **Album Information**:
   * Information about albums, including titles, release dates, artists, genres, and cover art.
4. **Audio Features**:
   * Datasets with audio analysis features such as tempo, key, danceability, loudness, and energy. You can obtain this data from services like the Spotify Web API.
5. **User Listening History**:
   * Data capturing user interactions, such as songs played, skipped tracks, likes, dislikes, and timestamps.
6. **User Profiles**:
   * User data, including usernames, email addresses, profile pictures, and preferences.
7. **Playlists**:
   * Data related to user-created playlists, including playlist titles, descriptions, and the songs they contain.
8. **Recommendation Feedback**:
   * Data on how users interact with recommendations, such as whether they liked, disliked, or added recommended songs to their playlists.
9. **Social Network Connections**:
   * Information about user connections, such as friends and followers, which can be used for social features like sharing playlists.
10. **Audio Files**:
    * Audio files for each song in the database, which can be streamed or downloaded by users.
11. **Genre Data**:
    * A dataset that associates songs and artists with specific genres to aid in music categorization and recommendation.
12. **Lyrics Data (Optional)**:
    * Lyrics for songs, which can be used for lyric display and search functionality.
13. **Historical Chart Data (Optional)**:
    * Data on historical music charts, such as Billboard charts, to provide users with information on song popularity over time.

It's important to note that acquiring and managing such datasets may involve licensing agreements and copyright considerations, especially when dealing with audio files and lyrics. Additionally, you may need to update and maintain the data to keep it current and relevant for users.

FROM KAGGLE:

**Song popularity**

<https://www.kaggle.com/datasets/yasserh/song-popularity-dataset>

<https://www.kaggle.com/datasets/mrmorj/dataset-of-songs-in-spotify> (might be useful for spotify like system)

# **Music Dataset : 1950 to 2019**

<https://www.kaggle.com/datasets/saurabhshahane/music-dataset-1950-to-2019>

<https://www.kaggle.com/datasets/srishreya/top-1000-songs-from-1920-to-2020/data>

**Location affect music taste**

<https://www.kaggle.com/code/hkapoor/music-trends-analysis-by-location/input>

<https://www.kaggle.com/datasets/danield2255/data-on-songs-from-billboard-19992019>

<https://www.kaggle.com/datasets/saurabhshahane/music-dataset-1950-to-2019>

<https://www.kaggle.com/datasets/pieca111/music-artists-popularity>