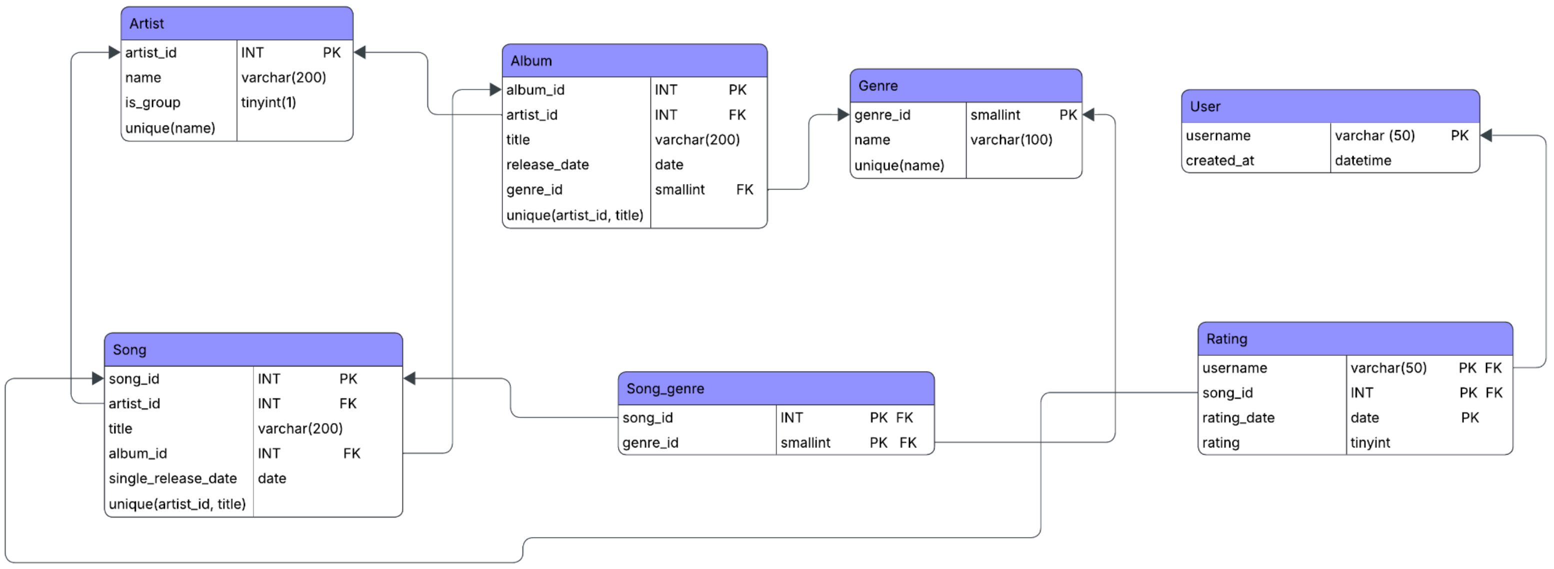

Relational Schema Explanation

This database is designed to model a simplified music-streaming service (similar to Spotify or Amazon Music). It stores artists, albums, songs, genres, users, and ratings, while minimizing redundancy through the use of primary keys and foreign keys.

Below, each table is described in terms of its purpose, columns and data types, primary key, foreign keys, and any uniqueness constraints. A final section explains how these choices satisfy the requirements in the assignment.

1. Artist

Purpose:
Represents either an individual performer or a group/band. Artists can release albums and single songs.

Columns and data types:

- **artist_id (primary key)** – Integer. Surrogate identifier for each artist.
- **name** – Variable-length string up to 200 characters. Stores the artist's name.
- **is_group** – Tiny integer (0 or 1) used as a Boolean flag to indicate whether the artist is a group/band (1) or an individual (0).

Unique constraints:

- **name** is unique: no two artists can share the same name.

Notes:
Using an integer primary key avoids repeating long artist names in other tables. The `is_group` flag allows us to distinguish individual artists from groups when running queries (e.g., “most prolific individual artists”).

2. Album

Purpose:
Represents an album released by a particular artist, on a specific date, in a single genre.

Columns and data types:

- **album_id (primary key)**– Integer. Surrogate identifier for each album.
- **artist_id** – Integer. References the artist who released the album.
- **title** – Variable-length string up to 200 characters. The album’s title.
- **release_date** – Date. The album’s release date.
- **genre_id** – Small integer. References the album’s single genre.

Foreign keys:

- **artist_id → Artist(artist_id)**
  Each album belongs to exactly one artist.
- **genre_id → Genre(genre_id)**
  Each album is classified into exactly one genre.

Unique constraints:

- The combination (**artist_id**, **title**) is unique. This allows the same album title to be reused by different artists, but prevents the same artist from having two different albums with the same title.

3. Genre

Purpose:
Represents a musical genre (e.g., Pop, Rock, R&B). Genres are predefined and reused for both albums and songs.

Columns and data types:

- **genre_id (primary key)**– Small integer. Surrogate identifier for each genre.
- **name** – Variable-length string up to 100 characters. The genre name.

Unique constraints:

- **name** is unique: no two genre rows share the same genre name.

Notes:
Using a small integer is sufficient because the number of genres is relatively limited.

4. Song

Purpose:
Represents an individual song. A song is performed by one artist and is either part of an album or a standalone single.

Columns and data types:

- **song_id (primary key)**– Integer. Surrogate identifier for each song.
- **artist_id** – Integer. References the artist who performs the song.
- **title** – Variable-length string up to 200 characters. The song's title.
- **album_id** – Integer, nullable. References the album that the song belongs to; left empty (NULL) for singles that are not part of any album.
- **single_release_date** – Date, nullable. Used as the release date for singles. For songs that are part of an album, this field is left empty (NULL), and the release date is taken from the album.

Foreign keys:

- **artist_id → Artist(artist_id)**
  Every song is performed by exactly one artist.

- **album_id → Album(album_id)** (nullable)
  If the song is part of an album, this links the song to that album. If it is a single, `album_id` is left NULL.

Unique constraints:

- The combination (**artist_id**, **title**) is unique. This enforces the rule that one artist cannot record two different songs with the same title, while still allowing different artists to have songs with the same title (covers).

Notes:

- The pair (**album_id**, **single_release_date**) is used to distinguish album tracks from singles:
  - Album tracks have a non-NULL `album_id` and a NULL `single_release_date`, and inherit their release date from the album.
  - Singles have a NULL `album_id` and a non-NULL `single_release_date`, so the song itself stores its release date.

5. Song_genre

Purpose:
Implements the many-to-many relationship between songs and genres. A song can belong to multiple genres, and a genre can contain many songs.

Columns and data types:

- **song_id** – Integer. References a song.

- **genre_id** – Small integer. References a genre.

Primary key:

- The combination (**song_id**, **genre_id**). This ensures that the same song cannot be linked to the same genre more than once.

Foreign keys:

- **song_id → Song(song_id)**
- **genre_id → Genre(genre_id)**

Notes:

- This table allows each song to be associated with one or more genres. The assignment specifies that every song must be in at least one genre; enforcing "at least one" is handled in the application logic when loading data.
- For songs that belong to an album, the genres assigned here are kept consistent with the album's genre (the application enforces that all tracks on an album share the album's genre).

6. User

Purpose:
Represents a user of the music service. Users can rate songs, but may also exist without any ratings.

Columns and data types:

- **username** – Variable-length string up to 50 characters. The user's unique identifier.
- **created_at** – Date-time value. Records when the user was created in the system.

Primary key:

- **username**

Notes:
The assignment states that users are uniquely identified by username, so we use `username` directly as the primary key (rather than introducing a numeric ID).

7. Rating

Purpose:
Represents a rating that a user gives to a specific song on a specific date.

Columns and data types:

- **username** – Variable-length string up to 50 characters. References the user who gave the rating.
- **song_id** – Integer. References the song being rated.
- **rating_date** – Date. The date on which the rating was made.
- **rating** – Tiny integer. Stores the rating value, which is restricted to the integers 1, 2, 3, 4, or 5.

Primary key:

- The combination (**username**, **song_id**, **rating_date**). This allows a user to rate the same song on different dates (for example, updating their opinion over time), but prevents multiple ratings for the same song by the same user on the same day.

Foreign keys:

- **username → User(username)**
- **song_id → Song(song_id)**

Notes:
The rating's numeric type and the documented constraint ensure that only the allowed values 1–5 are stored.

How the Schema Meets the Assignment Requirements

1. **Artists**
    - Each artist (individual or group) is stored in the **Artist** table and uniquely identified by the **name** column. A numeric **artist_id** is used as the primary key to avoid repeating large strings. The **is_group** flag distinguishes individual artists from bands, which supports queries like "most prolific individual artists."
2. **Albums**

- Each album is represented in the **Album** table, linked to a single artist via **artist_id** and having a **release_date**.

- The constraint that “an album name is not unique, but the combination of album name and artist name is unique” is enforced by making the pair (**artist_id**, **title**) unique.

- Every album has exactly one genre via **genre_id**, which ties to the **Genre** table.

3. **Songs and Singles**

   - The **Song** table stores each song’s **title**, **artist_id**, and either an **album_id** (for album tracks) or a **single_release_date** (for singles not part of any album).

   - The uniqueness of song titles per artist is enforced by the unique combination (**artist_id**, **title**). Different artists may still share the same title, allowing covers.

   - For album tracks, the release date is taken from the associated album. For singles, the release date is stored directly in **single_release_date**, matching the assignment’s description.

4. **Genres and Multi-genre Songs**

   - All genres are stored once in the **Genre** table.

   - The many-to-many relationship between songs and genres is modeled by the **Song_genre** table.

   - This structure allows a song to belong to one or more genres, as required. The application logic ensures that every song has at least one genre, and that album tracks’ genres align with their album’s single genre.

5. **Users**

   - Users are uniquely identified by **username** in the **User** table, exactly as specified.

   - Users can exist without any associated ratings, because ratings are stored in a separate table.

6. **Ratings**

- The **Rating** table ties together **User** and **Song** through foreign keys, and includes both a **rating_date** and a numeric **rating** value.

- The rating is stored as a tiny integer and constrained conceptually to the values 1 through 5, satisfying the requirement that ratings be limited to 1–5.

- Because the primary key includes **username**, **song_id**, and **rating_date**, each rating is tied to a specific user, song, and date, and users may rate the same song again on a different day if needed.

7. **Minimizing Redundancy**

    - Artists, albums, songs, genres, users, and ratings are each stored once in their own tables.

    - Relationships (such as song–genre and user–song ratings) are modeled with separate tables that use foreign keys rather than repeating descriptive information.

    - This design avoids duplication of names, titles, and genre labels, and keeps the database in a normalized form that is well suited for querying and for implementing the required Python functions later.