ERD - Cardinality

Introduction

In database design, relationships between tables are crucial for organizing and structuring data efficiently. We'll explore three types of relationships: one-to-one (1:1), one-to-many (1:many), and many-to-many (many-to-many).

1. One-to-One (1:1) Relationship

A one-to-one relationship exists when each record in Table A corresponds to exactly one record in Table B, and vice versa.

Example: A person and their passport

Person
person_id
name
date_of_birth

Passport
passport_id
person_id
issue_date
expiry_date

CREATE statements:

SQL

```sql
CREATE TABLE Person (
    person_id INT PRIMARY KEY,
    name VARCHAR(100),
    date_of_birth DATE
);

CREATE TABLE Passport (
    passport_id INT PRIMARY KEY,
    person_id INT UNIQUE,
    issue_date DATE,
    expiry_date DATE,
    FOREIGN KEY (person_id) REFERENCES Person(person_id)
);
```

Note: The `UNIQUE` constraint on `person_id` in the Passport table ensures the one-to-one relationship.

2. One-to-Many (1:many) Relationship

A one-to-many relationship exists when a record in Table A can be associated with multiple records in Table B, but each record in Table B is associated with only one record in Table A.

Example: An author and their books

Author
author_id
name
nationality

Book
book_id
title
author_id
publish_date

CREATE statements:

```sql
CREATE TABLE Author (
    author_id INT PRIMARY KEY,
    name VARCHAR(100),
    nationality VARCHAR(50)
);

CREATE TABLE Book (
    book_id INT PRIMARY KEY,
    title VARCHAR(200),
    author_id INT,
    publish_date DATE,
    FOREIGN KEY (author_id) REFERENCES Author(author_id)
);
```

3. Many-to-Many (many-to-many) Relationship

A many-to-many relationship exists when multiple records in Table A can be associated with multiple records in Table B, and vice versa. This relationship typically requires a junction table.

Example: Students and courses

Student
student_id

Student
name
email

Course
course_id
title
credits

Enrollment
student_id
course_id
semester

CREATE statements:

```sql
CREATE TABLE Student (
    student_id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100)
);

CREATE TABLE Course (
    course_id INT PRIMARY KEY,
    title VARCHAR(200),
    credits INT
);

CREATE TABLE Enrollment (
    student_id INT,
    course_id INT,
    semester VARCHAR(20),
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES Student(student_id),
    FOREIGN KEY (course_id) REFERENCES Course(course_id)
);
```

Note: The Enrollment table serves as a junction table, connecting students and courses. The primary key is a combination of `student_id` and `course_id` to ensure unique enrollment records.

Conclusion

Understanding these relationships is essential for effective database design:

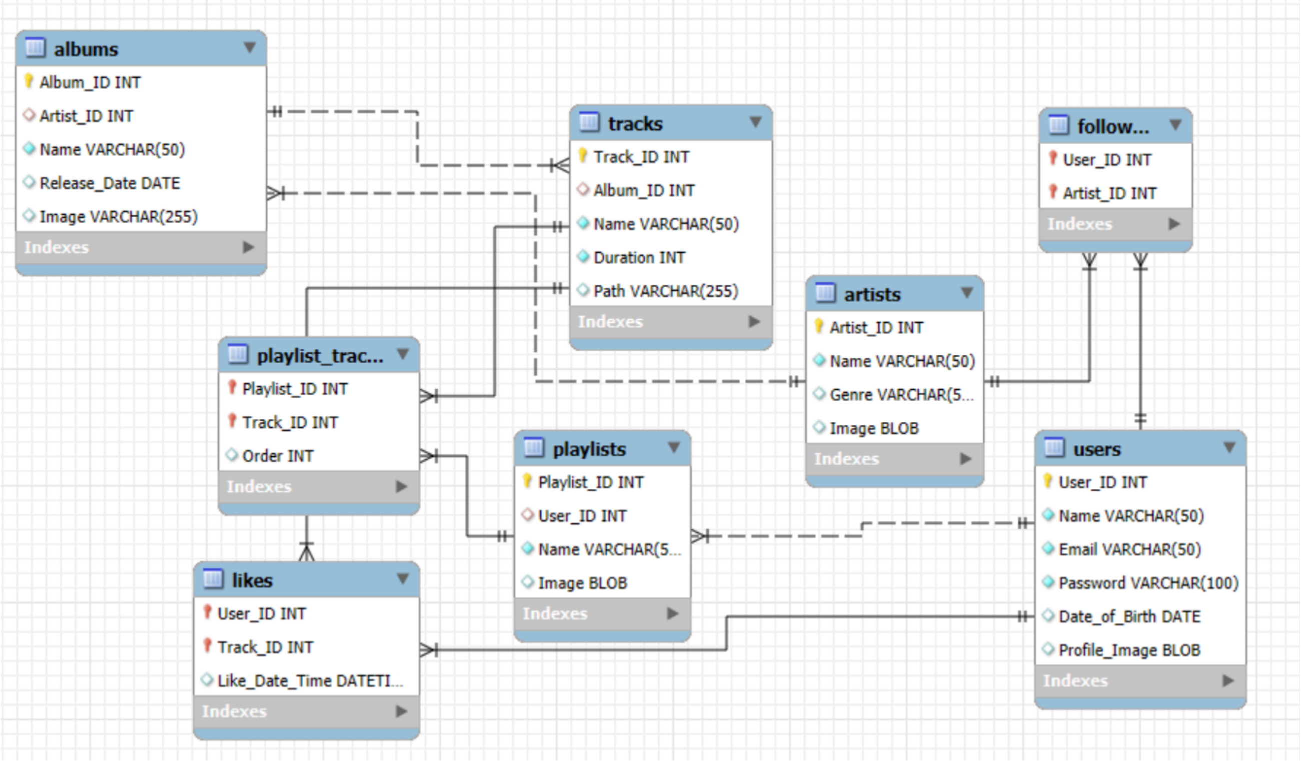
- One-to-One (1:1) relationships connect two tables with a unique relationship between records.
- One-to-Many (1:many) relationships allow a record in one table to be associated with multiple records in another table.
- Many-to-Many (many-to-many) relationships require a junction table to connect multiple records from both tables.

Proper implementation of these relationships ensures data integrity, reduces redundancy, and allows for efficient querying and data management in MySQL databases.

Spotify Schema

```sql
CREATE TABLE users (
User_ID INT AUTO_INCREMENT PRIMARY KEY,
Name VARCHAR(50) NOT NULL,
Email VARCHAR(50) NOT NULL UNIQUE,
Password VARCHAR(100) NOT NULL,
Date_of_Birth DATE,
Profile_Image Blob
);

CREATE TABLE artists (
Artist_ID INT AUTO_INCREMENT PRIMARY KEY,
Name VARCHAR(50) NOT NULL,
Genre VARCHAR(50),
Image Blob
);


CREATE TABLE albums (
Album_ID INT AUTO_INCREMENT PRIMARY KEY,
Artist_ID INT,
Name VARCHAR(50) NOT NULL,
Release_Date DATE,
Image VARCHAR(255),
FOREIGN KEY (Artist_ID) REFERENCES Artists(Artist_ID)
);

CREATE TABLE tracks (
Track_ID INT AUTO_INCREMENT PRIMARY KEY,
Album_ID INT,
Name VARCHAR(50) NOT NULL,
Duration INT NOT NULL,
Path VARCHAR(255),
FOREIGN KEY (Album_ID) REFERENCES Albums(Album_ID)
);


CREATE TABLE playlists (
Playlist_ID INT AUTO_INCREMENT PRIMARY KEY,
User_ID INT,
Name VARCHAR(50) NOT NULL,
Image Blob,
FOREIGN KEY (User_ID) REFERENCES Users(User_ID)
);
```

```sql
CREATE TABLE playlist_tracks (
Playlist_ID INT,
Track_ID INT,
`Order` INT,
PRIMARY KEY (Playlist_ID, Track_ID),
FOREIGN KEY (Playlist_ID) REFERENCES Playlists(Playlist_ID),
FOREIGN KEY (Track_ID) REFERENCES Tracks(Track_ID)
);

CREATE TABLE followers (
User_ID INT,
Artist_ID INT,
PRIMARY KEY (User_ID, Artist_ID),
FOREIGN KEY (User_ID) REFERENCES Users(User_ID),
FOREIGN KEY (Artist_ID) REFERENCES Artists(Artist_ID)
);


CREATE TABLE likes (
User_ID INT,
Track_ID INT,
Like_Date_Time DATETIME,
PRIMARY KEY (User_ID, Track_ID),
FOREIGN KEY (User_ID) REFERENCES Users(User_ID),
FOREIGN KEY (Track_ID) REFERENCES Tracks(Track_ID)
);
```



Relationships

The ERD shows several types of relationships:

1. **One-to-Many**:
   - An artist can have many albums (Artist_ID in the albums table)
   - An album can have many tracks (Album_ID in the tracks table)
   - A user can create many playlists (User_ID in the playlists table)
2. **Many-to-Many** (implemented via junction tables):
   - Playlists can contain many tracks, and tracks can belong to many playlists (via playlist_tracks)
   - Users can like many tracks, and tracks can be liked by many users (via likes)
   - Users can follow many artists, and artists can be followed by many users (via follows)