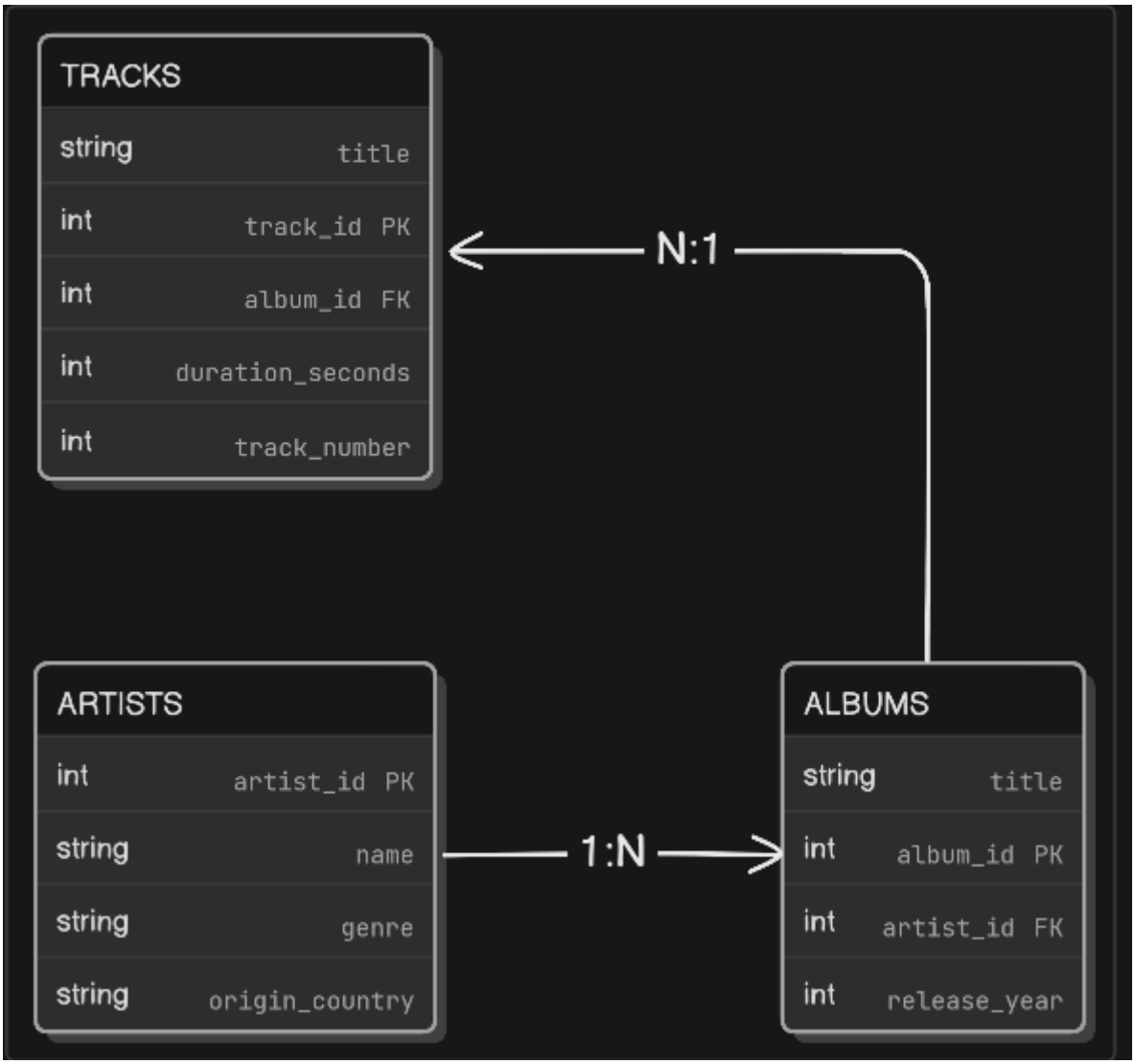

Diagrammet visar en hierarkisk struktur med en-till-många-relationer (1:N).

Artists 1:N Albums: En artist kan ha skapat flera album, men i denna modell tillhör ett album specifikt en artist (via artist_id).

Albums 1:N Tracks: Ett album innehåller flera låtar, men en specifik instans av en låt tillhör ett album (via album_id).

Förklaringar och motiveringar

Databasdesign (Normalisering): Databasen är designad enligt normaliseringsprinciper för att undvika dataredundans (upprepning).

Istället för att spara artistens namn på varje rad i låt-tabellen, sparar vi artisten *en* gång i artists-tabellen.

Jag länkar sedan ihop tabellerna med foreign keys (främmande nycklar). Detta gör databasen mer effektiv och lättare att uppdatera (om en artist byter namn behöver vi bara ändra på ett ställe).

Datatyper:

INT IDENTITY(1,1): Används för Primary Keys (artist_id,). Detta garanterar att varje rad får ett unikt ID automatiskt, vilket SQL Server hanterar.

VARCHAR(X): Används för text (namn, titlar). Vi sätter en gräns (t.ex. 100 eller 150 tecken) för att optimera lagringen, men tillräckligt långt för att namn ska få plats.

ON DELETE CASCADE: Denna regel lades till på våra Foreign Keys. Om vi tar bort en artist (t.ex. The Beatles), kommer SQL Server automatiskt att ta bort alla deras album och låtar.

SKAPA TABELLER

```
CREATE TABLE artists (
    artist_id INT IDENTITY(1,1) PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    genre VARCHAR(50),
    origin_country VARCHAR(50)
);

CREATE TABLE albums (
    album_id INT IDENTITY(1,1) PRIMARY KEY,
    artist_id INT,
    title VARCHAR(150) NOT NULL,
    release_year INT,
    FOREIGN KEY (artist_id) REFERENCES artists(artist_id) ON DELETE CASCADE
);

CREATE TABLE tracks (
    track_id INT IDENTITY(1,1) PRIMARY KEY,
    album_id INT,
    title VARCHAR(150) NOT NULL,
```

```
    duration_seconds INT,
    track_number INT,
    FOREIGN KEY (album_id) REFERENCES albums(album_id) ON DELETE CASCADE
);

LÄGG TILL DATA
INSERT INTO artists (name, genre, origin_country) VALUES
('The Beatles', 'Rock', 'UK'),
('Daft Punk', 'Electronic', 'France'),
('Miles Davis', 'Jazz', 'USA');

INSERT INTO albums (artist_id, title, release_year) VALUES
(1, 'Abbey Road', 1969),
(1, 'Revolver', 1966),
(2, 'Discovery', 2001),
(2, 'Random Access Memories', 2013),
(3, 'Kind of Blue', 1959);

INSERT INTO tracks (album_id, title, duration_seconds, track_number) VALUES
(1, 'Come Together', 259, 1),
(1, 'Something', 182, 2),
(1, 'Maxwell"s Silver Hammer', 207, 3),
(2, 'Taxman', 158, 1),
(2, 'Eleanor Rigby', 126, 2),
(3, 'One More Time', 320, 1),
(3, 'Aerodynamic', 207, 2),
(3, 'Digital Love', 298, 3),
(4, 'Give Life Back to Music', 274, 1),
(4, 'Get Lucky', 369, 8),
```

(5, 'So What', 562, 1),

(5, 'Freddie Freeloader', 586, 2),

(5, 'Blue in Green', 337, 3);

LINQ-jämförelser

Att hämta alla artister från UK

SQL: SELECT * FROM artists WHERE origin_country = 'UK';

LINQ: var ukArtists = context.Artists .Where(a => a.OriginCountry == "UK") .ToList();

Exempel 2

Hämta album med artistens namn

SQL: SELECT albums.title, artists.name FROM albums JOIN artists ON albums.artist_id = artists.artist_id;

LINQ: var albumsWithArtist = context.Albums .Include(a => a.Artist) // Motsvarar JOIN .Select(a => new { AlbumTitle = a.Title, ArtistName = a.Artist.Name }) .ToList();

Reflektion: Som vi ser här är det ganska mindre kod och mer lättläst än LINQ, dock är ju kanske mer lättläst för oss c# utvecklare då vi pluggar det, men jag har läst lite om sql innan så tycker det är smidigare.

Säkerhetsdelen

Risk: Om vi låter användare skriva in data (t.ex. söka efter en artist) och vi klistrar in den texten direkt i vårt SQL-kommando, kan en hackare skriva in kod som raderar databasen.

Lösning: Använd Parameteriserade frågor (i SQL) eller Entity Framework/LINQ (som gör detta automatiskt). Aldrig konkatenera strängar ("SELECT * FROM artists WHERE name = '" + userInput + "'" är farligt).

Behörigheter (Least Privilege):

Risk: Att applikationen loggar in i databasen som (System Admin) eller "root". Om applikationen hackas har inkräktaren full kontroll över hela servern.

Lösning: Skapa en specifik databasanvändare för applikationen som bara har rättigheter att göra SELECT, INSERT, UPDATE och DELETE enbart i musiclibrary-databasen, inget annat. (Skapa en admin user!)

Kort reflektion:

Vad gick bra:

Att förstå den logiska strukturen (Artist -> Album -> Låt) kändes naturligt.

Att skapa tabellerna gick smidigt när syntaxen väl var korrekt för SQL Server.

Vad var svårt:

Skillnaden i syntax mellan olika SQL-dialekter (t.ex. AUTO_INCREMENT i MySQL vs IDENTITY i SQL Server). Det orsakade felmeddelanden i början som var förvirrande.

Att hålla reda på Foreign Key-constraints; man måste skapa (och ta bort) tabeller i rätt ordning (Artists måste skapas före Albums).

Vad kan förbättras:

Databasen saknar index (förutom på Primary Keys). Om vi hade miljoner låtar skulle sökningar på "Song Title" gå långsamt. Jag borde lägga till en CREATE INDEX på namn och titlar.

Jag kunde även lagt till en "Many-to-Many"-tabell för artister, eftersom vissa låtar har "Featured artists" (samarbeten), vilket nuvarande struktur inte hanterar perfekt.