

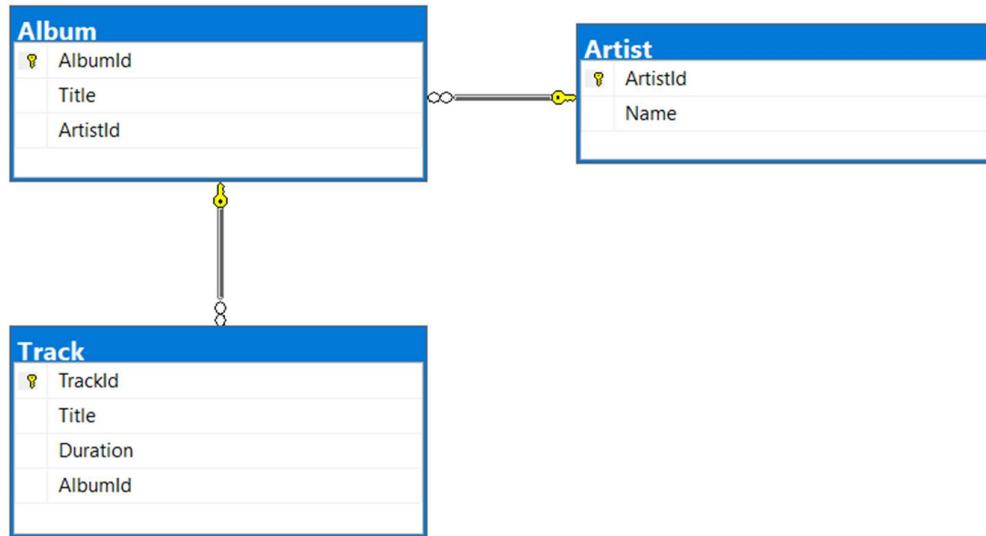
Musikdatabas

Ehsan Sistani

Sammanfattning

Väljer att skapa en musikdatabas med Entiteter som, Artist, Album, och Track. Attributer väljs enligt diagrammet nedan. Musikdatabasen innehåller relationskopplingar som Artist-Album som då är 1:N relation. Album-Track relationen får också 1:N. Diagrammet nedan visar relationerna markerade med nyckel N har ersatts med symbolen ∞ . I kolumnen Artist får Attributet ArtistId med Primary key (PK) vilket är en unik identifierare för varje artist som också förhindrar dubletter av ArtistId. I tabellen Album finns en FK (ArtistId) som kopplar varje Album till en Artist via ArtistId. För att kunna deklarera en FK behöver man antigen ange en explicit constraint (FK_Album_Artist) i sql-tabellen, om det inte anges skapas en sådant automatiskt. En FK (AlbumId) till finns mellan Track-Album, vilket menas att varje unik låt tillhör en specifik album.

ER-diagram



Entiteter och Attributer

Artist

- ArtistId (PK) – unik identifierare för varje artist
- Name – artistens namn

Album

- AlbumId (PK) – unik identifierare för varje album
- Title – albumets titel
- ArtistId (FK) – referens till den artist som äger albumet

Track

- TrackId (PK) – unik identifierare för varje spår
- Title – spårets titel
- Duration – spårets längd (t.ex. i sekunder eller minuter)
- AlbumId (FK) – referens till albumet spåret tillhör

Förklaring till attributer

- Artist: Har bara ArtistId och Name.
- Album: Kopplas till en artist via ArtistId (FK).
- Track: Kopplas till ett album via AlbumId (FK).
- Constraints:

PRIMARY KEY för unika ID:n.

NOT NULL för obligatoriska fält.

CHECK (Duration > 0) för att förhindra spår med längd 0 eller negativ.

FOREIGN KEY för att säkerställa relationerna.

Relationer

Artist–Album (1:N), Album–Track (1:N)

Artist → Album: En **artist** kan ha **många album**. Det betyder att relationen från *Artist* till *Album* är **1:N** (en till många).

Album → Track: Ett album kan innehålla **många spår**. Därför är relationen från albumets perspektiv **1:N** (en till många).

SQL-kommandos

lägger in sql-koderna, create_tables, insert_data, select_basic, select_join, updates.

Create_tables.sql

```
--skapa en databas  
--create database Musikdatabas  
--börja använda databasen  
use Musikdatabas  
-- Skapa tabellen Artist  
CREATE TABLE Artist (  
    ArtistId INT PRIMARY KEY NOT NULL, -- Unik identifierare för varje artist  
    Name NVARCHAR(100) NOT NULL      -- Artistens namn  
);  
-- Skapa tabellen Album  
CREATE TABLE Album (  
    AlbumId INT PRIMARY KEY NOT NULL, -- Unik identifierare för varje album  
    Title NVARCHAR(200) NOT NULL,     -- Albumets titel  
    ArtistId INT NOT NULL,           -- Referens till Artist  
    CONSTRAINT FK_Album_Artist FOREIGN KEY (ArtistId)  
        REFERENCES Artist(ArtistId)  
);  
-- Skapa tabellen Track  
CREATE TABLE Track (  
    TrackId INT PRIMARY KEY NOT NULL, -- Unik identifierare för varje spår  
    Title NVARCHAR(200) NOT NULL,     -- Spårets titel  
    Duration INT NOT NULL CHECK (Duration > 0), -- Spårets längd i sekunder/minuter  
    AlbumId INT NOT NULL,           -- Referens till Album  
    CONSTRAINT FK_Track_Album FOREIGN KEY (AlbumId)  
        REFERENCES Album(AlbumId)  
);
```

insert_data.sql

```
-- 1. Visa alla låtar som är längre än 200 sekunder
```

```
SELECT Title, Duration
```

```
FROM Track
```

```
WHERE Duration > 200;
```

```
-- 2. Visa alla låtar från album 1 sorterade efter längd
```

```
SELECT Title, Duration
```

```
FROM Track
```

```
WHERE AlbumId = 1
```

```
ORDER BY Duration DESC;
```

```
-- 3. Visa alla artister vars namn börjar med 'R'
```

```
SELECT ArtistId, Name
```

```
FROM Artist
```

```
WHERE Name LIKE 'R%';
```

```
-- 4. Visa alla album sorterade alfabetiskt
```

```
SELECT AlbumId, Title
```

```
FROM Album
```

```
ORDER BY Title ASC;
```

```
-- 5. Antal låtar per album
```

```
SELECT AlbumId, COUNT(*) AS NumberOfTracks
```

```
FROM Track
```

```
GROUP BY AlbumId;
```

```
-- 6. Total speltid per album
```

```
SELECT AlbumId, SUM(Duration) AS TotalDuration
```

```
FROM Track
```

```
GROUP BY AlbumId;
```

```
-- 7. Visa alla låtar vars titel innehåller ordet 'Song'
```

```
SELECT TrackId, Title  
FROM Track  
WHERE Title LIKE '%Song%';
```

```
-- 8.Visa artister sorterade efter namn i fallande ordning
```

```
SELECT ArtistId, Name  
FROM Artist  
ORDER BY Name DESC;
```

```
-- 9. Genomsnittlig låtlängd per album
```

```
SELECT AlbumId, AVG(Duration) AS AverageDuration  
FROM Track  
GROUP BY AlbumId;  
select_basic.sql  
-- 1. Visa alla låtar som är längre än 200 sekunder  
SELECT Title, Duration  
FROM Track  
WHERE Duration > 200;
```

```
-- 2.Visa alla låtar från album 1 sorterade efter längd
```

```
SELECT Title, Duration  
FROM Track  
WHERE AlbumId = 1  
ORDER BY Duration DESC;
```

```
-- 3. Visa alla artister vars namn börjar med 'R'
```

```
SELECT ArtistId, Name
FROM Artist
WHERE Name LIKE 'R%';

-- 4. Visa alla album sorterade alfabetiskt
SELECT AlbumId, Title
FROM Album
ORDER BY Title ASC;

-- 5. Antal låtar per album
SELECT AlbumId, COUNT(*) AS NumberOfTracks
FROM Track
GROUP BY AlbumId;

-- 6. Total speltid per album
SELECT AlbumId, SUM(Duration) AS TotalDuration
FROM Track
GROUP BY AlbumId;

-- 7. Visa alla låtar vars titel innehåller ordet 'Song'
SELECT TrackId, Title
FROM Track
WHERE Title LIKE '%Song%';

-- 8. Visa artister sorterade efter namn i fallande ordning
SELECT ArtistId, Name
FROM Artist
ORDER BY Name DESC;

-- 9. Genomsnittlig låtlängd per album
SELECT AlbumId, AVG(Duration) AS AverageDuration
```

```
FROM Track
```

```
GROUP BY AlbumId;
```

```
USE Musikdatabas;
```

select_join.sql

```
-- 1. Visa alla låtar med albumtitel och artistnamn
```

```
SELECT
```

```
Track.Title,
```

```
Track.Duration,
```

```
Album.Title,
```

```
Artist.Name
```

```
FROM Track
```

```
JOIN Album ON Track.AlbumId = Album.AlbumId
```

```
JOIN Artist ON Album.ArtistId = Artist.ArtistId;
```

```
-- 2. Visa alla album och antal låtar per album
```

```
SELECT Album.Title,
```

```
Artist.Name,
```

```
COUNT(Track.TrackId) AS NumberOfTracks
```

```
FROM Album
```

```
JOIN Artist ON Album.ArtistId = Artist.ArtistId
```

```
LEFT JOIN Track ON Album.AlbumId = Track.AlbumId
```

```
GROUP BY Album.Title, Artist.Name;
```

updates.sql

```
-- 1. Uppdatera artistnamnet för 'Lonely Singer' till 'Solo Star'
```

```
UPDATE Artist
```

```
SET Name = 'Solo Star'
```

```
WHERE ArtistId = 3;
```

```
-- 2. Uppdatera titeln på albumet 'Heavy Roads' till 'Heavy Highways'
```

```
UPDATE Album
```

```
SET Title = 'Heavy Highways'
```

```
WHERE AlbumId = 4;
```

```
-- 3. (Extra exempel) Uppdatera längden på låten 'Future Sound' till 260 sekunder
```

```
UPDATE Track
```

```
SET Duration = 260
```

```
WHERE Title = 'Future Sound';
```

deletes.sql

```
USE Musikdatabas;
```

```
-- 1. Ta bort en artist (Lonely Singer) som inte har några album
```

```
DELETE FROM Artist
```

```
WHERE ArtistId = 3;
```

```
-- 2. Ta bort en låt med titeln 'Extra Song 5'
```

```
DELETE FROM Track
```

```
WHERE Title = 'Extra Song 5';
```

LINQ-jämförelser

```
--Jämför SQL och LINQ
```

```
--1. Filtrera låtar längre än 200 sekunder
```

```
SELECT Title, Duration
```

```
FROM Track
```

```
WHERE Duration > 200;
```

```
--linq
```

```
var result = Tracks  
.Where(x => x.Duration > 200)  
.Select(x => new { x.Title, x.Duration })  
.ToList();
```

--2. Visa alla album sorterade alfabetiskt

```
SELECT AlbumId, Title  
FROM Album  
ORDER BY Title ASC;
```

--linq

```
var result = Albums  
.OrderBy(x => x.Title)  
.Select(x => new { x.AlbumId, x.Title })  
.ToList();
```

--3. Antal låtar per album

```
SELECT AlbumId, COUNT(*) AS NumberOfTracks  
FROM Track  
GROUP BY AlbumId;
```

--linq

```
var result = Tracks  
.GroupBy(x => x.AlbumId)  
.Select(g => new { AlbumId = g.Key, NumberOfTracks = g.Count() })  
.ToList();
```

Säkerhet:

- **Varför säker åtkomst till databaser är viktigt**

Man ska kunna förhindra att data hamnar fel i händer. Utan säker åtkomst kan vem som helst komma åt databas-informationen vilket är inte önskevärt. Det betyder att datan blir tillgängligt och obehöriga kan få tillgång till datan. För att förhindra detta kan man t.ex. låta backend läsa/skriva, begränsa användare till läsa/skriva. Exempel vanliga användare får läsa sin egen information medans admin får göra lite mer på så sätt minska buggar och fel i systemet. Man minskar risken för attacker som riktas mot databaser och gör det svårare för angripare att lyckas komma in i systemet.

- **Vad authentication och authorization betyder**

Ett sätt att kunna identifiera sig användare man **authentication**, för att kunna visa vem man är, t.ex via Bank-id.

När man har loggat in i ett system då bestäms vad du får göra (**authorization**), t.ex admin kan lägga till/ta bort användare vanlig användare. En vanlig användare har bara tillgång till sin egen profil, en gäst kan bara läsa.

- **Hur man skyddar känslig data i ett backendprojekt**

Man kan använda fler lager av säkerhet som kryptering (t.ex https), Hashning, säkra hemligheter(skydd av API-nycklar), rättigheter(authentication och authentication), validering, övervakning, backup. Då man kombinerar allt ovan får man ett backend som är både robust och säker.