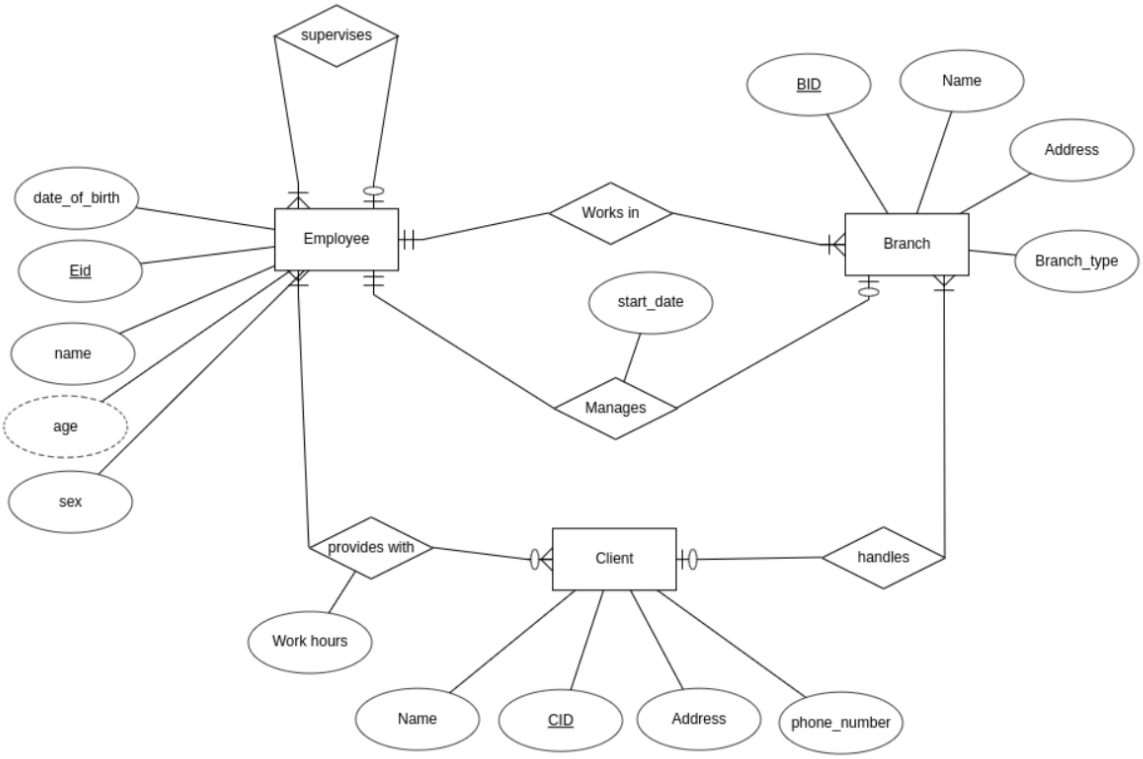
Problem 1: MIDF

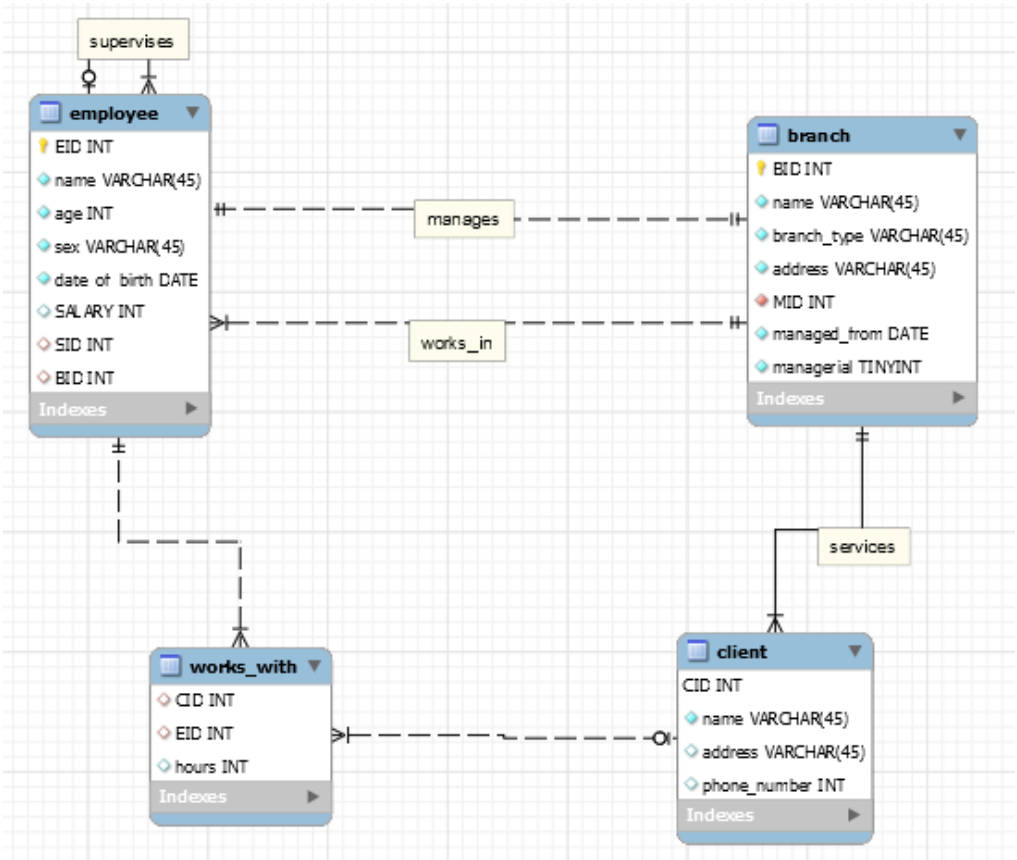
Problem Statement

Founder of Meeseeks Interior Design farm (MID in short) Mr. Meeseeks wants to utilize digital database system to store his company's data instead of files and cabinets based analog database they were maintaining so far. According to Mr. Meeseeks this is how the farm operates.
MID makes money by providing the interior designing services to its clients, which are mainly homeowners. MID stores name, address, phone number of its clients for future support and so on. MID has many branches spread around the country. Each branch has a name, address. Every branch also has an employee who manages the branch. The date when the employee starts managing the branch is kept in record to measure his performance. Not all branches provide service to the clients. Some of them are just there for other managerial tasks. Such as the head branch where Mr. Meeseeks himself go to monitor the operations of his farm. Every branch has employees. After all employees are the workforce of MID. In other words, there is no branch without employees. Each employee's name, salary, age, sex, date of birth is kept in records. Employees also has employee IDs. Each employee in MID is only associated with one branch. An employee can also act as a supervisor of multiple other employees. Every employee has to have exactly one supervisor. Employees work with the clients controlled by the branch. Multiple employees can work with the same client. How many hours an employee worked with a client is also kept in record.

Solution

The entities in this problem are, Employee, Branch and Client.
There is a one to many relationships between branch and employee. Each employee will have a branch_id that signifies which branch they work in.
Each branch is managed by only one employee therefore the manages relationship is one to one. There will be a manager_id, and starting_date in branch.
One employee supervises many other employees so there is a one to many relationships between employee to employee.
Branch handles clients when a client comes to a branch. One branch can handle many clients so there is a one to many relationships between client and branch. Clients are optional to the branches because not all branch handles clients that's why there is an optional '0' marking in client's side of relationship. Client table would have a handler_branch attribute to record which branch they went to.
There is many to many relationships between client and employee as one employee can provide service to many clients and also many employees can work for one client. We need to make a separate table to record which client worked with which client and for how many hours. Therefore, the table will hold the customer_id, employee_id and work hours in each row.


supervises
BID
Name
Address
date_of_birth
Works in
Branch_type
Eid
Employee
Branch
start_date
name
Manages
age
sex
provides with
Client
handles
Work hours
Name
CID
Address
phone_number


supervises
employee
EID INT
name VARCHAR(45)
age INT
sex VARCHAR(45)
date of birth DATE
SALARY INT
SID INT
BID INT
Indexes
manages
works_in
branch
BID INT
name VARCHAR(45)
branch_type VARCHAR(45)
address VARCHAR(45)
MID INT
managed_from DATE
managerial TINYINT
Indexes
services
works_with
CID INT
EID INT
hours INT
Indexes
client
CID INT
name VARCHAR(45)
address VARCHAR(45)
phone_number INT
Indexes

The corresponding SQL:

```
CREATE SCHEMA IF NOT EXISTS `assignment1` DEFAULT CHARACTER SET utf8 ;
USE `assignment1` ;

CREATE TABLE IF NOT EXISTS `assignment1`.`branch` (
`BID` INT NOT NULL AUTO_INCREMENT,
`name` VARCHAR(45) NOT NULL,
`branch_type` VARCHAR(45) NOT NULL,
`address` VARCHAR(45) NOT NULL,
`manager_ID` INT NOT NULL,
`managed_from` DATE NOT NULL,
PRIMARY KEY (`BID`),
INDEX ` manager_ID _idx` (` manager_ID ` ASC) VISIBLE,
CONSTRAINT ` manager_ID `
FOREIGN KEY (` manager_ID `)
REFERENCES `assignment1`.`employee` (`EID`)
ON DELETE NO ACTION
ON UPDATE NO ACTION);


CREATE TABLE IF NOT EXISTS `assignment1`.`employee` (
`EID` INT NOT NULL,
`name` VARCHAR(45) NOT NULL,
`age` INT NOT NULL,
`sex` VARCHAR(45) NOT NULL,
`date_of_birth` DATE NOT NULL,
`SALARY` INT NULL,
`supervisor_ID` INT NULL,
`BID` INT NULL,
PRIMARY KEY (`EID`),
INDEX `MID_idx` (` supervisor_ID ` ASC) VISIBLE,
INDEX `BID_idx` (`BID` ASC) VISIBLE,
CONSTRAINT `SID`
FOREIGN KEY (` supervisor_ID `)
REFERENCES `assignment1`.`employee` (`EID`)
ON DELETE NO ACTION
```

```
ON UPDATE NO ACTION,
CONSTRAINT `BID`
FOREIGN KEY (`BID`)
REFERENCES `assignment1`.`branch` (`BID`)
ON DELETE NO ACTION
ON UPDATE NO ACTION);


CREATE TABLE IF NOT EXISTS `assignment1`.`client` (
`CID` INT NOT NULL AUTO_INCREMENT,
`name` VARCHAR(45) NOT NULL,
`address` VARCHAR(45) NULL,
`phone_number` INT NULL,
PRIMARY KEY (`CID`),
CONSTRAINT `branch`
FOREIGN KEY (`handler_branch_ID`)
REFERENCES `assignment1`.`branch` (` handler_branch_ID `)
ON DELETE NO ACTION
ON UPDATE NO ACTION);


CREATE TABLE IF NOT EXISTS `assignment1`.`works_with` (
`CID` INT NULL,
`EID` INT NULL,
`hours_worked` INT NULL,
INDEX `EID_idx` (`EID` ASC) VISIBLE,
INDEX `client_idx` (`CID` ASC) VISIBLE,
CONSTRAINT `client`
FOREIGN KEY (`CID`)
REFERENCES `assignment1`.`client` (`CID`)
ON DELETE NO ACTION
ON UPDATE NO ACTION,
CONSTRAINT `emp`
FOREIGN KEY (`EID`)
REFERENCES `assignment1`.`employee` (`EID`)
ON DELETE NO ACTION
ON UPDATE NO ACTION
```

Problem 2: Album Collection

Problem Statement

A friend is interested in keeping track of information about his album collection. He is not concerned about whether or not the albums are CDs, tapes, LPs, etc. Also, assume that he does not have any compilation albums—that is, each album has songs from a single band. For each album, he wants to store which band recorded the album, the title, the year, and the chronology (e.g. this is the 4^{th} album for that band). He also wants to store the songs, including title, length, track number, and writer(s). Of course, if two bands record the same song, they might have different track numbers and lengths. For each band (group or individual), he also wants to store the names of all of the band members. For each band member, he needs their first and last names, and country of origin. Consider both band members and songwriters as musicians.

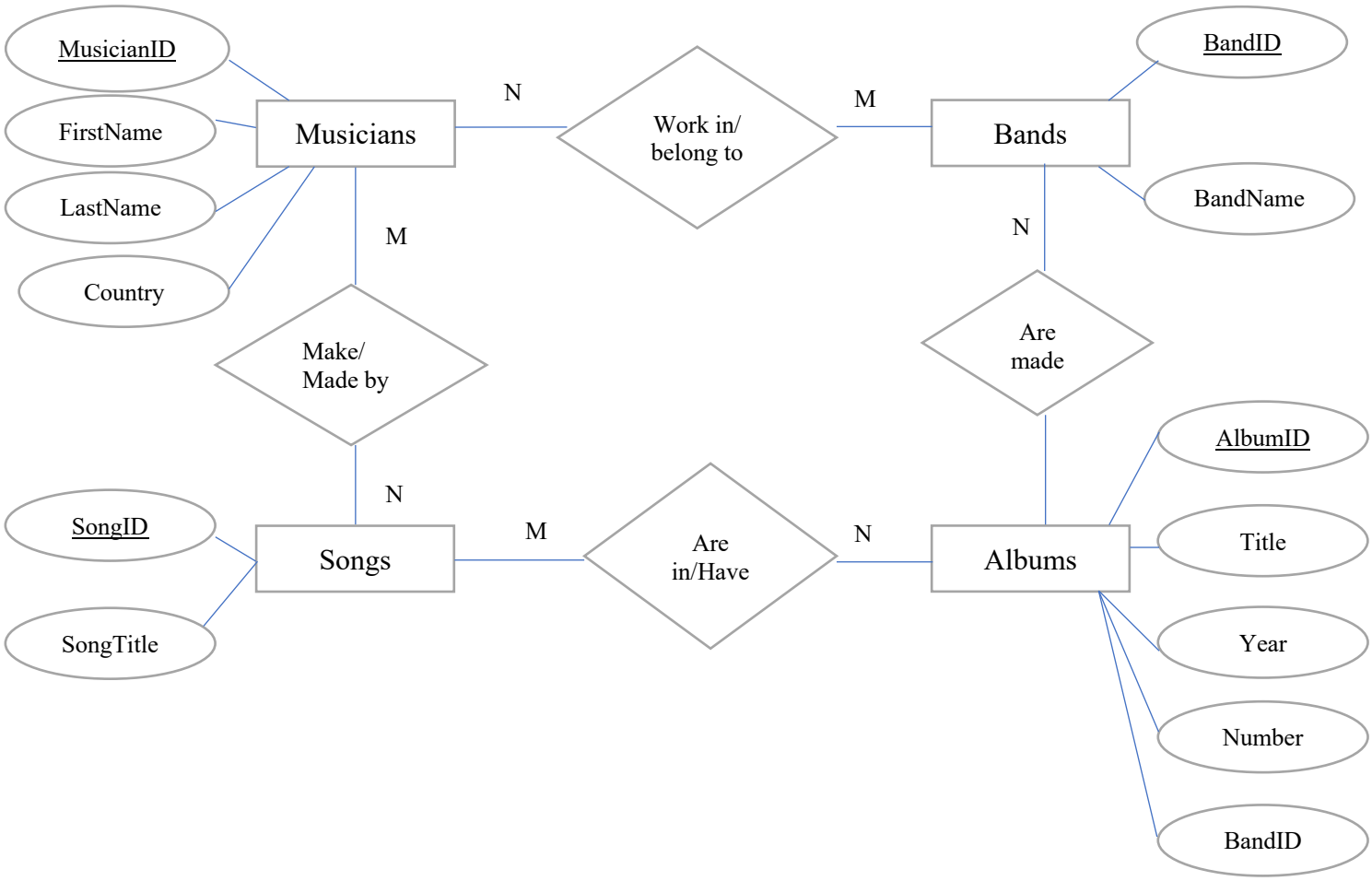
Construct a database to store the required information so he can do things like list all of the information for an album, list all songs written by a musician, etc.

Solution

What we know is:

- The obvious entities are **Musicians**, **Bands**, **Albums**, and **Songs**.
- There is a one-to-many relationship between **Albums** and **Bands**, which will require adding *BandID* to **Albums**.
- For each **Musician**, we need to store *MusicianFirstName*, *MusicianLastName*, and *MusicianCountry.* Since these may not be unique, we add *MusicianID* as a primary key.
- For each **Band**, we need to store the **BandName**, and *BandID* for a primary key.
- For each **Album**, we need to store *AlbumTitle*, *AlbumYear*, *AlbumNumber* (for chronology), *BandID* from **Bands**, and *AlbumID* for a primary key.
- For each **Song**, we need *SongTitle* and *SongID* as primary key.

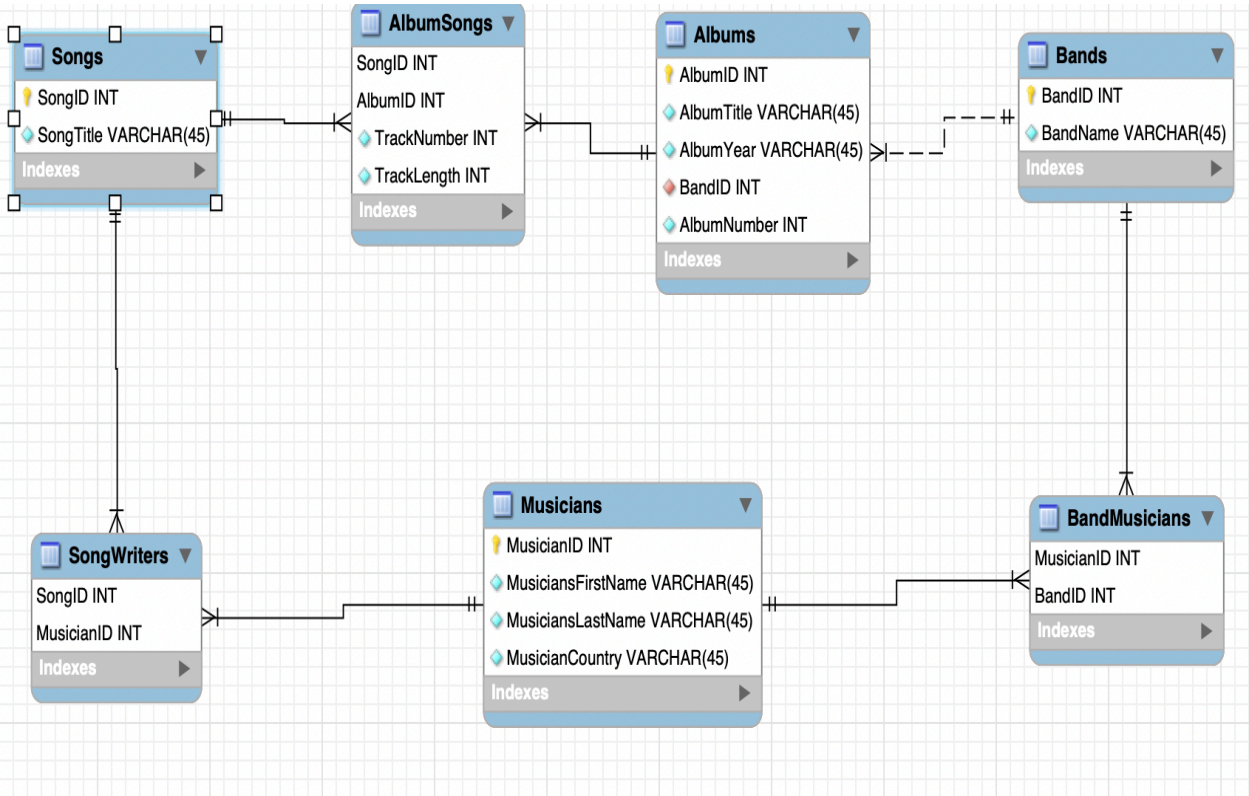
Given these facts, we can construct our primary entity-relationship (ER) model:
Recall that the *N*s and *M*s in the diagram indicate the cardinality of the relationships. For instance, **Musicians** and **Bands** have a many-to-many relationship, whereas **Albums** and **Bands** have a one-to-many relationship.



To convert the ER model into tables:

- We need to deal with many-to-many relationships. Each of these involve two tables. For each many-to-many relationship, we add a new table with a name relating to both tables that are related, include as foreign keys the primary keys of the two related tables, and make that the primary key of the new table. Also, add any attributes that are part of the relationship, and an attribute to indicate order, if necessary.
- There is a many-to-many relationship between **Musicians** and **Bands**, and **Musicians** and **Songs**, and **Albums** and **Songs**.Therfore,
- We need a **BandMusicians** table with *BandID* and *MusicianID* to
- We need a **SongWriters** table with *SongID* and *MusicianID*.
- We need a **AlbumSongs** table with *SongID*, *AlbumID*, *TrackLength*, and *TrackNumber*. The *TrackLength* needs to be stored here rather than in the **Songs** table since different recordings could have different lengths.

From this, we can construct the following tables



And SQL commands accordingly:

```
-- Schema mydb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
USE `mydb` ;

-- -----------------------------------------------------
-- Table `mydb`.`Musicians`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Musicians` (
  `MusicianID` INT NOT NULL,
  `MusiciansFirstName` VARCHAR(45) NOT NULL,
  `MusiciansLastName` VARCHAR(45) NOT NULL,
  `MusicianCountry` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`MusicianID`),
  UNIQUE INDEX `MusicianID_UNIQUE` (`MusicianID` ASC) VISIBLE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Bands`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`Bands` (
  `BandID` INT NOT NULL AUTO_INCREMENT,
  `BandName` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`BandID`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Songs`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Songs` (
  `SongID` INT NOT NULL AUTO_INCREMENT,
  `SongTitle` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`SongID`),
  UNIQUE INDEX `SongID_UNIQUE` (`SongID` ASC) VISIBLE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Albums`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Albums` (
  `AlbumID` INT NOT NULL AUTO_INCREMENT,
  `AlbumTitle` VARCHAR(45) NOT NULL,
  `AlbumYear` VARCHAR(45) NOT NULL,
  `BandID` INT NOT NULL,
  `AlbumNumber` INT NOT NULL,
  PRIMARY KEY (`AlbumID`),
  INDEX `BandID_idx` (`BandID` ASC) VISIBLE,
  CONSTRAINT `BandID`
    FOREIGN KEY (`BandID`)
    REFERENCES `mydb`.`Bands` (`BandID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`AlbumSongs`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`AlbumSongs` (
  `SongID` INT NOT NULL,
  `AlbumID` INT NOT NULL,
  `TrackNumber` INT NOT NULL DEFAULT 0,
  `TrackLength` INT NOT NULL DEFAULT 0,
  PRIMARY KEY (`SongID`, `AlbumID`),
  INDEX `AlbumID_idx` (`AlbumID` ASC) VISIBLE,
  CONSTRAINT `SongID`
    FOREIGN KEY (`SongID`)
```

```
    REFERENCES `mydb`.`Songs` (`SongID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `AlbumID`
    FOREIGN KEY (`AlbumID`)
    REFERENCES `mydb`.`Albums` (`AlbumID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`SongWriters`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`SongWriters` (
  `SongID` INT NOT NULL AUTO_INCREMENT,
  `MusicianID` INT NOT NULL,
  PRIMARY KEY (`SongID`, `MusicianID`),
  INDEX `MusicianID_idx` (`MusicianID` ASC) VISIBLE,
  CONSTRAINT `SongID`
    FOREIGN KEY (`SongID`)
    REFERENCES `mydb`.`Songs` (`SongID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `MusicianID`
    FOREIGN KEY (`MusicianID`)
    REFERENCES `mydb`.`Musicians` (`MusicianID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`BandMusicians`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`BandMusicians` (
  `MusicianID` INT NOT NULL,
  `BandID` INT NOT NULL,
  PRIMARY KEY (`MusicianID`, `BandID`),
  INDEX `BandID_idx` (`BandID` ASC) VISIBLE,
  CONSTRAINT `MusicianID`
    FOREIGN KEY (`MusicianID`)
    REFERENCES `mydb`.`Musicians` (`MusicianID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `BandID`
    FOREIGN KEY (`BandID`)
    REFERENCES `mydb`.`Bands` (`BandID`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```