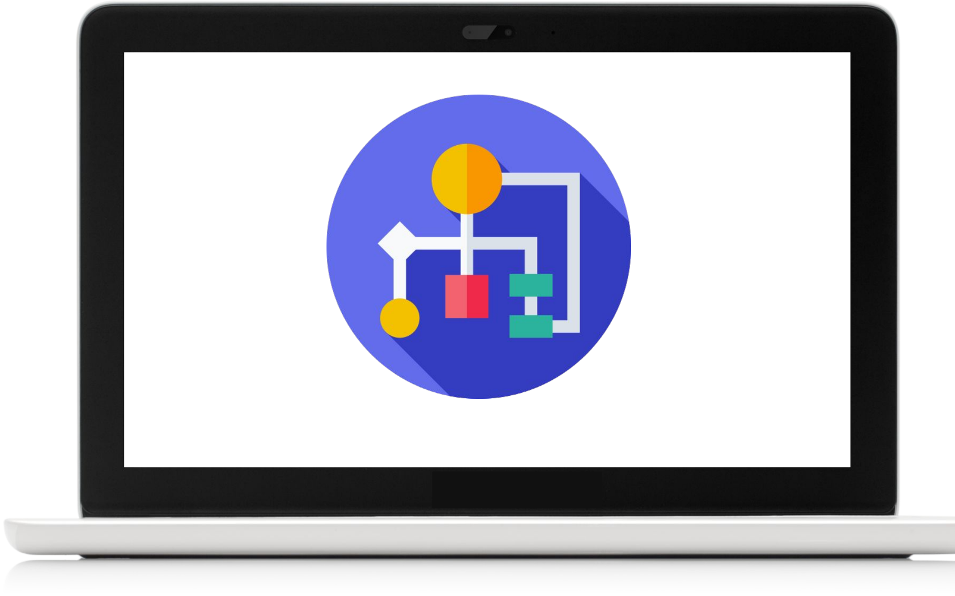
ER Diagrams

Please go through material for these topics. Complete the reading, exercises, and any videos linked. If the instructions ask to turn in any exercises, please do so through slack to your instructor.

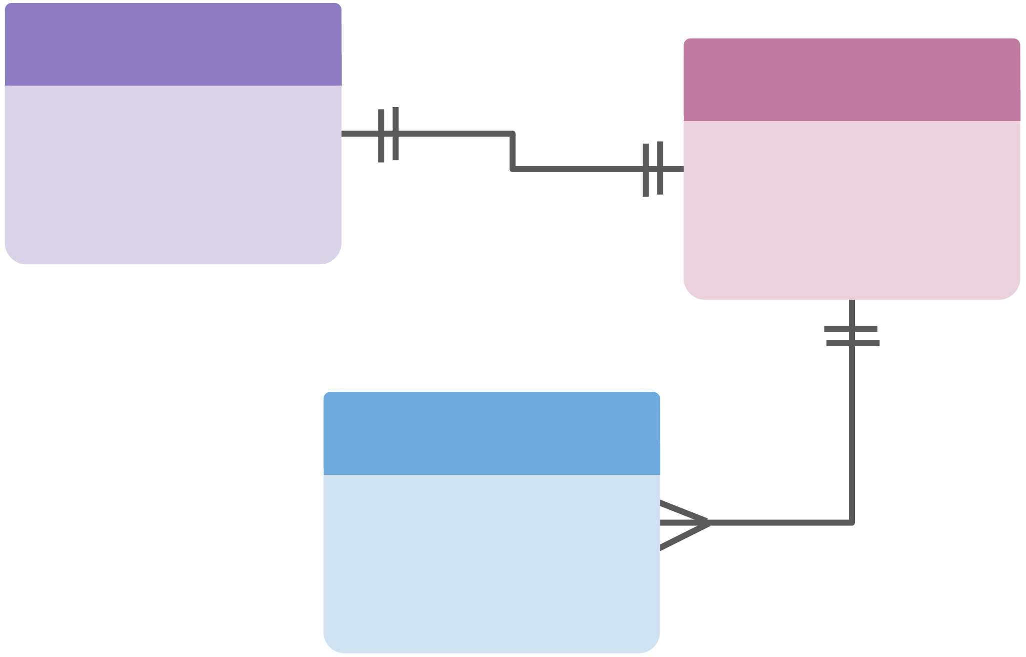


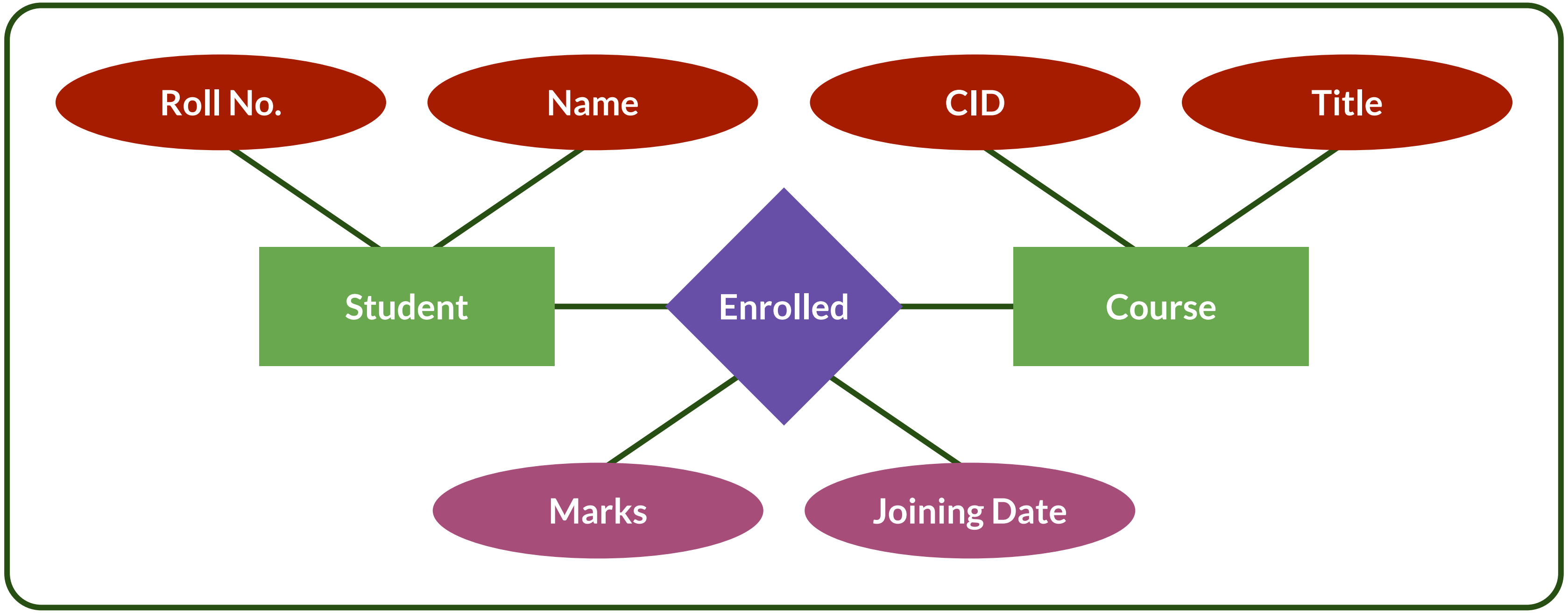
Cognixia™

Outline

1. Introduction to ER Diagrams
2. Rules for an ER
3. Creating an ER
4. Exercise
5. Open Book Quiz

ER Diagrams





Entity Relationship Diagrams can be very basic overviews of the structure of a database. You may have previously seen a diagram like above, but ER Diagrams can get more detailed to give exact look at the data stored and the table structures in a database.

Why Use ER Diagrams?

- Helpful in giving a ***full overview of a database*** and understanding the relationships between tables
- When you are given access to a database and there is no documentation on the structure of the database, an ER Diagram can be generated
- Creating ER Diagrams before writing your database script can ***avoid future design issues and need to recreate database*** because of it (especially if that database is already being used by an outside application)
- First we are going to look at how ER diagrams are designed and then how we can generate an ER diagram through MySQL from existing databases

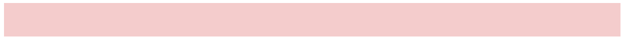
Outline Single Table

- ➔ A single entity or table will be represented by a box with the **name at the top and the columns listed below**
- ➔ You will detail any ***keys*** (primary or foreign keys) that exists within the columns and the ***data types*** for each column
- ➔ Not common, but can add constraints to an ER as well

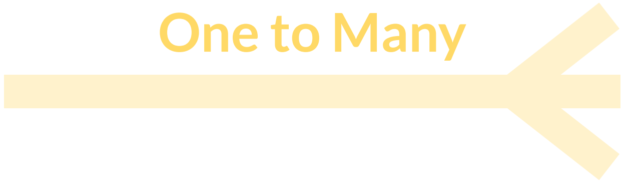
Employee		
PK	id	int
	name	varchar(255)
	dob	date
	salary	double
FK	dept_id	int

Relationships

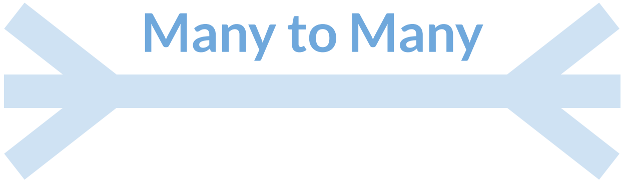
One to One



One to Many



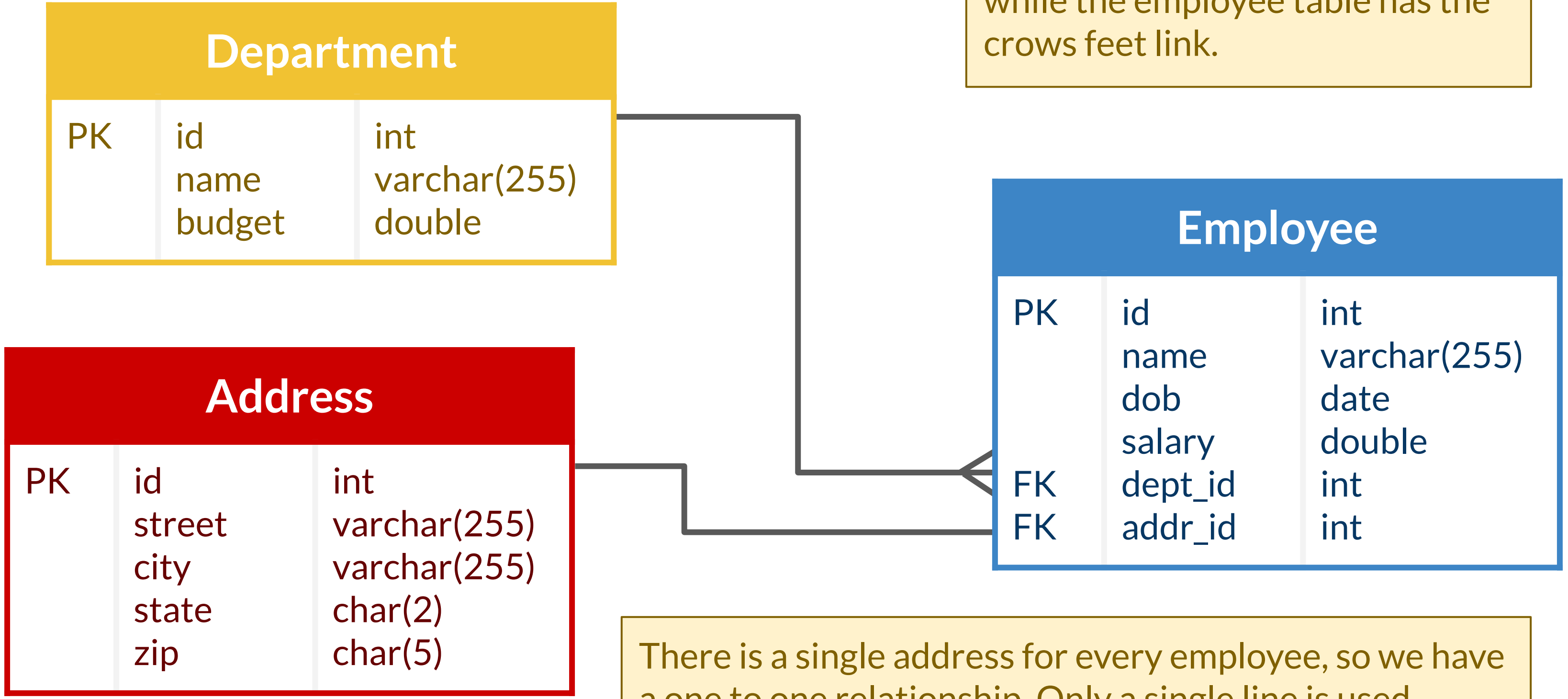
Many to Many



- ➔ To show the 1-to-1, 1-to-Many, and Many-to-Many relationships, we'll connect each entity with a line
- ➔ If it ends in a **single line**, that entity is the ***one*** in the relationship
- ➔ If it has **three lines (crows feet)**, it is the ***many*** in the relationship

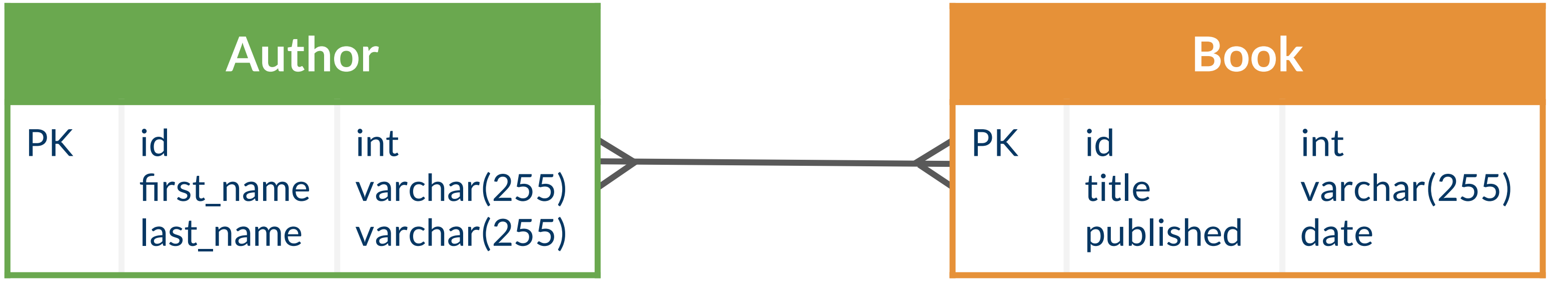
In order to build a database that tracks employees in a company we can set up the following ER Diagram.

A single department has many employees so we have a one to many relationship. The department will have a single line while the employee table has the crows feet link.



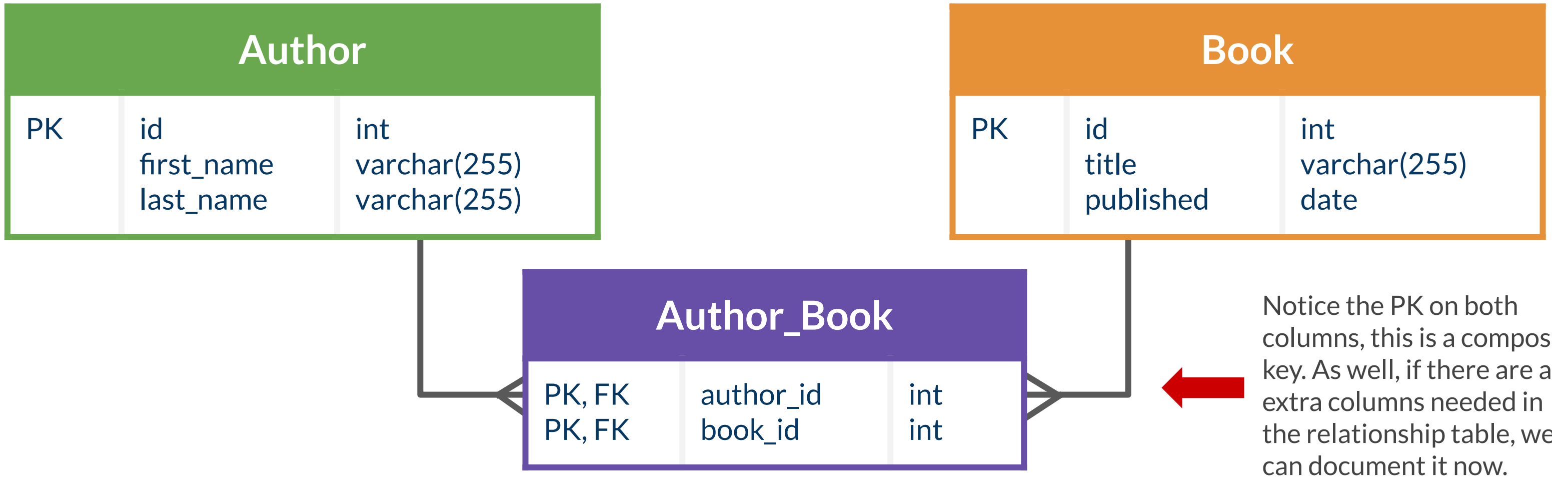
There is a single address for every employee, so we have a one to one relationship. Only a single line is used.

Many-to-Many Relationships



- ➔ In a many to many relationship, you can use a double crows feet line
- ➔ However ***not often used in most ER Diagrams*** as it does not show off the joining or relationship table
- ➔ However, if you are creating a quick diagram while brainstorming ideas, it is fine to do something like above where we want to show that a book can have many authors and an author can write many books

Many-to-Many Relationships



Author

PK	id	int
	first_name	varchar(255)
	last_name	varchar(255)

Book

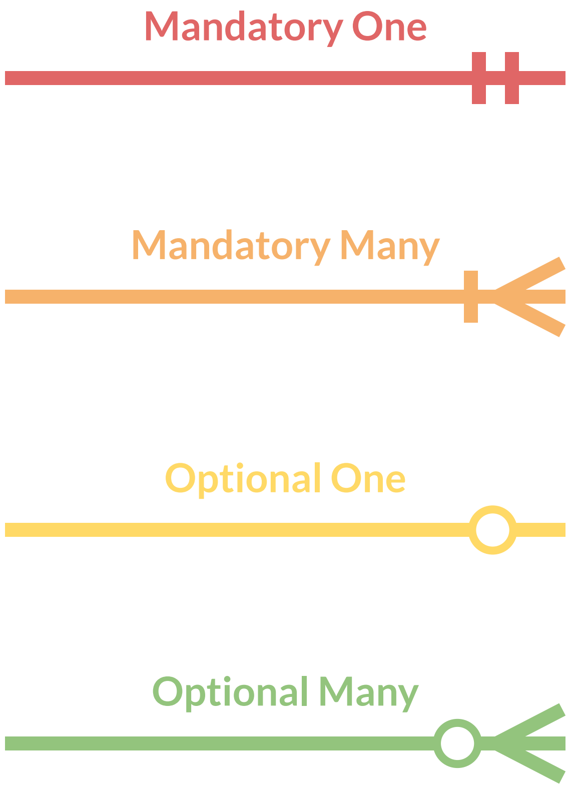
PK	id	int
	title	varchar(255)
	published	date

Author_Book

PK, FK	author_id	int
PK, FK	book_id	int

Notice the PK on both columns, this is a composite key. As well, if there are any extra columns needed in the relationship table, we can document it now.

- ➔ **Use two one-to-many relationship lines** between each entity and the relationship/joining table
- ➔ This is the way you'll see it represented in nearly all ER Diagrams you come across



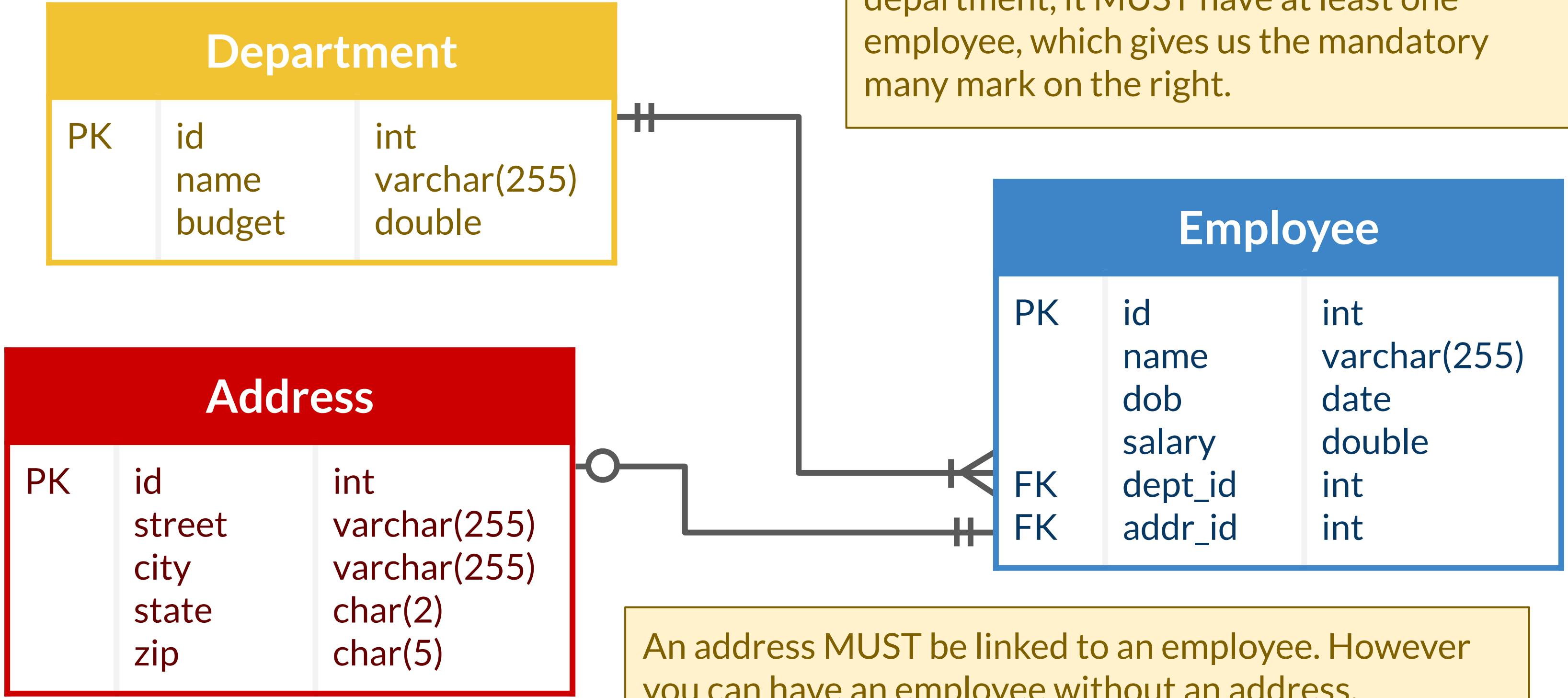
Mandatory & Optional Relationship

➔ A ***single or double cross line*** indicates a **mandatory relationship**, an instance or record of the entity must exist in the relationship (at least one or exactly one)

➔ A ***circle on the line*** indicates an **optional relationship**, an instance or record of the entity doesn’t need to exist in the relationship (zero or more)

When we update our ER Diagram for the employee database, we include mandatory and optional marks.

In an employee and department relationship, records on both sides must exist. If there is an employee they MUST have a department, hence the mandatory one marks. If there is a department, it MUST have at least one employee, which gives us the mandatory many mark on the right.



Department

PK	id	int
	name	varchar(255)
	budget	double

Address

PK	id	int
	street	varchar(255)
	city	varchar(255)
	state	char(2)
	zip	char(5)

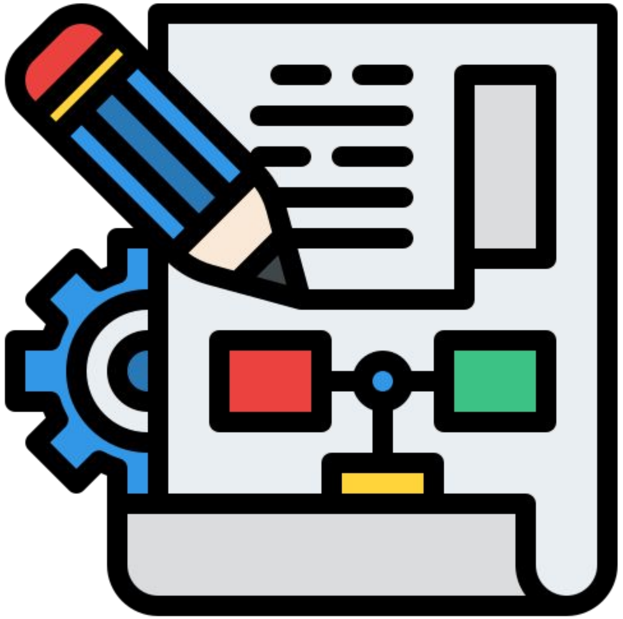
Employee

PK	id	int
	name	varchar(255)
	dob	date
	salary	double
FK	dept_id	int
FK	addr_id	int

An address MUST be linked to an employee. However you can have an employee without an address.

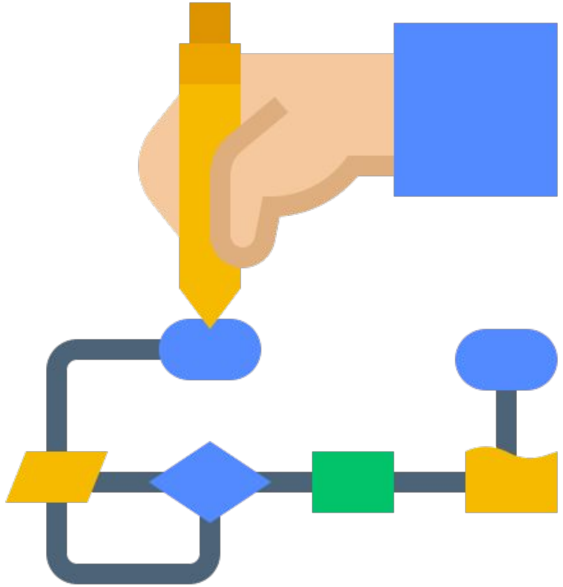
Database Planning

- ➔ ***In the future, the databases you work with will already exist***, you will not be designing an SQL database
- ➔ However as you create future projects through this course, you will need to design your own databases to track data needed for the applications you build
- ➔ As such, it is good to know how to create your own ER Diagrams to **plan your databases first before creating (then build it out), before writing whatever application you are using the database for**



Tools for Creating an ER Diagram

- ➔ You can create a diagram using the diagramming tool from google **Draw.io** (can even save it to your google drive)
  - ◆ https://app.diagrams.net/
- ➔ **Lucidchart** is another free tool that is very popular for creating all types of diagrams (you will need to create an account, but it is free)
  - ◆ https://www.lucidchart.com/pages/
- ➔ You can also search for other diagramming tools online or draw it yourself on a piece of paper



Demo: Create ER for Music Streaming

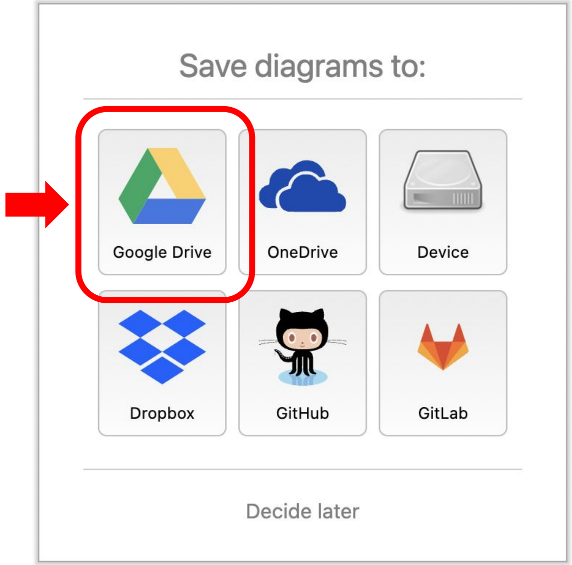
➔ For this demo, we will be using **Draw.io**

➔ Let’s assume we are tasked with creating an application for a music streaming site that will need a database, check the requirements below

Scenario: You are building an application for a music streaming company. They need to store information for artists names, album names, album release date, genre, song name, song length. What kind of ER do you build to model your data?

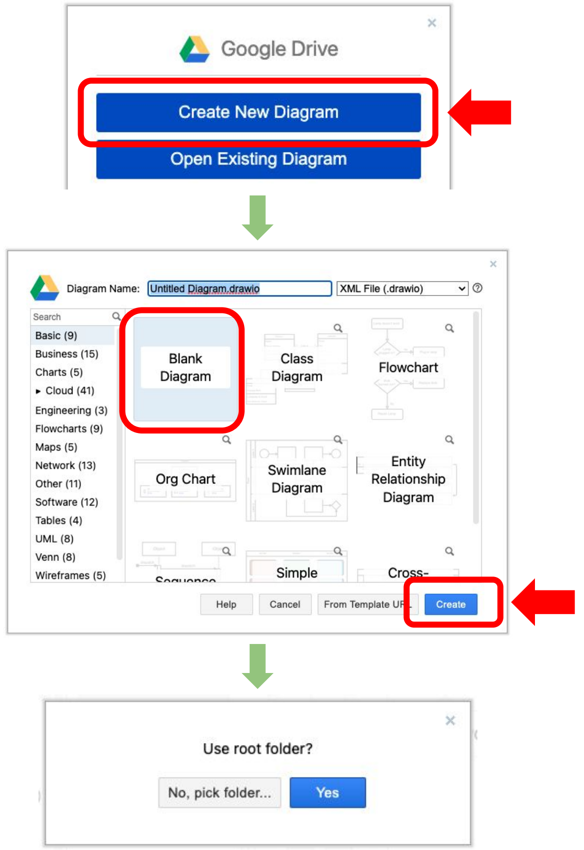
Open up Your ER Tool

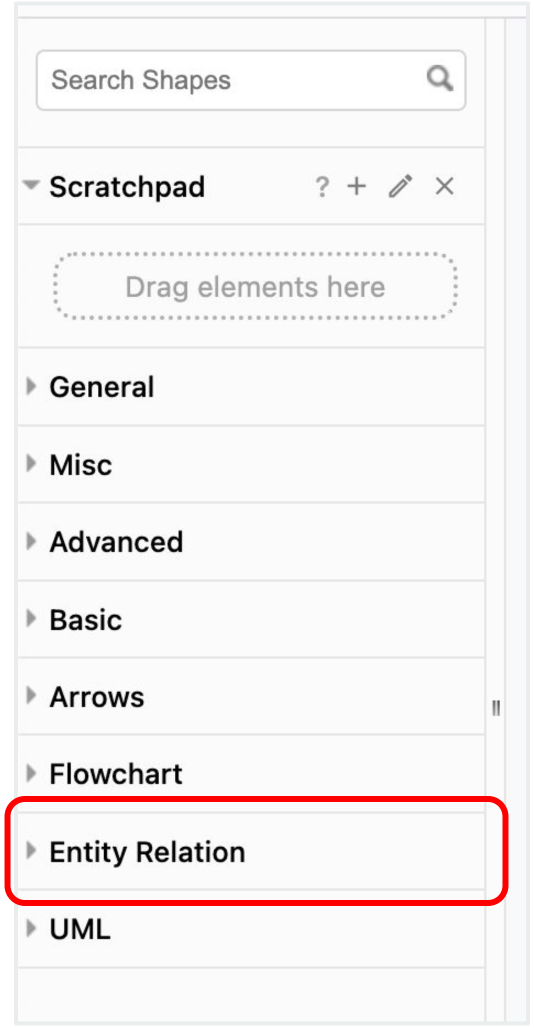
- ➔ Open up your diagram tool: https://app.diagrams.net/
- ➔ You will get an option of where you want to save the file, if you have a google account, select the google drive option
- ➔ You may be asked to allow permissions for use of this application, just accept

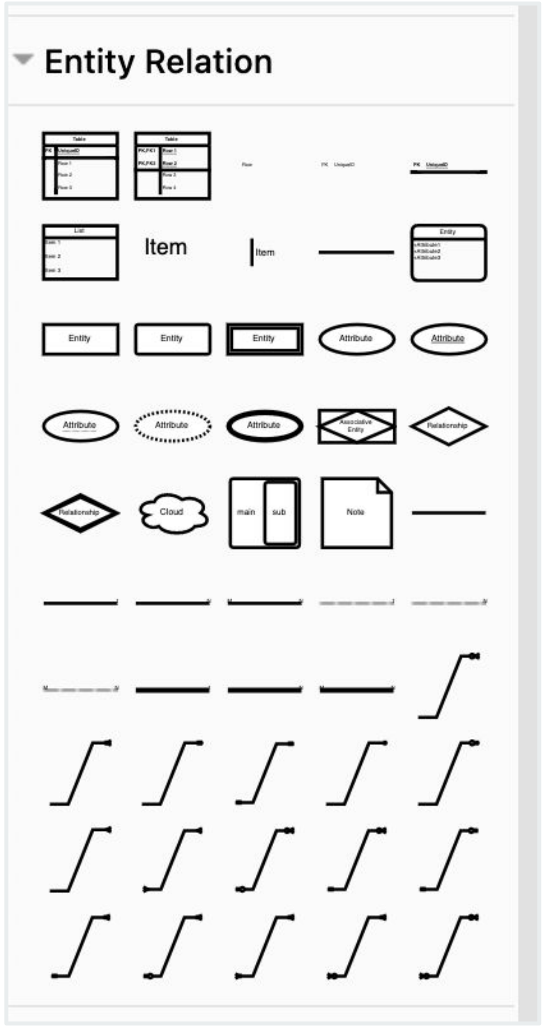


Open up Your ER Tool

- ➔ Click the option to create a new diagram
- ➔ Then when the window pops up on which diagram to create, just select the blank diagram option
- ➔ If saving to your google drive, you'll see an option to select which folder you want to save your diagram in
    - ◆ If you select yes to use root folder it will just save diagram in your main google drive folder

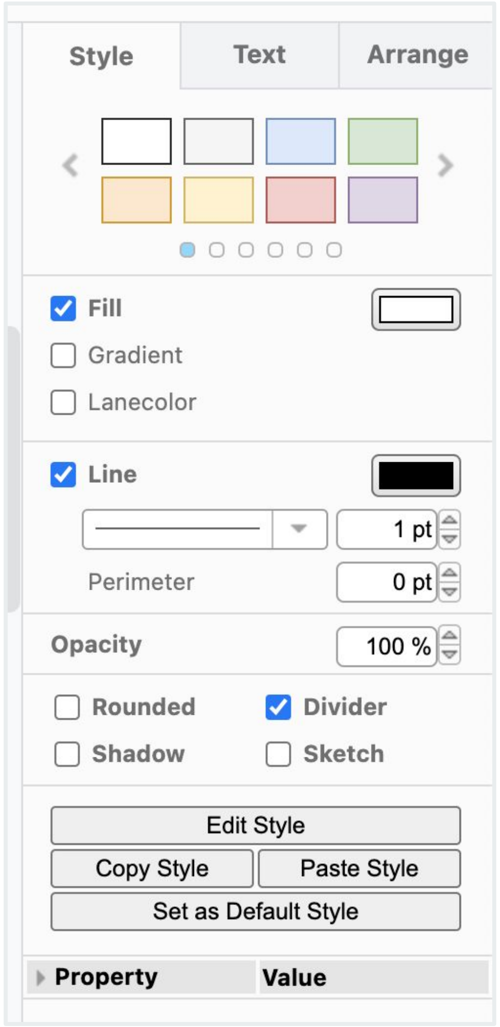




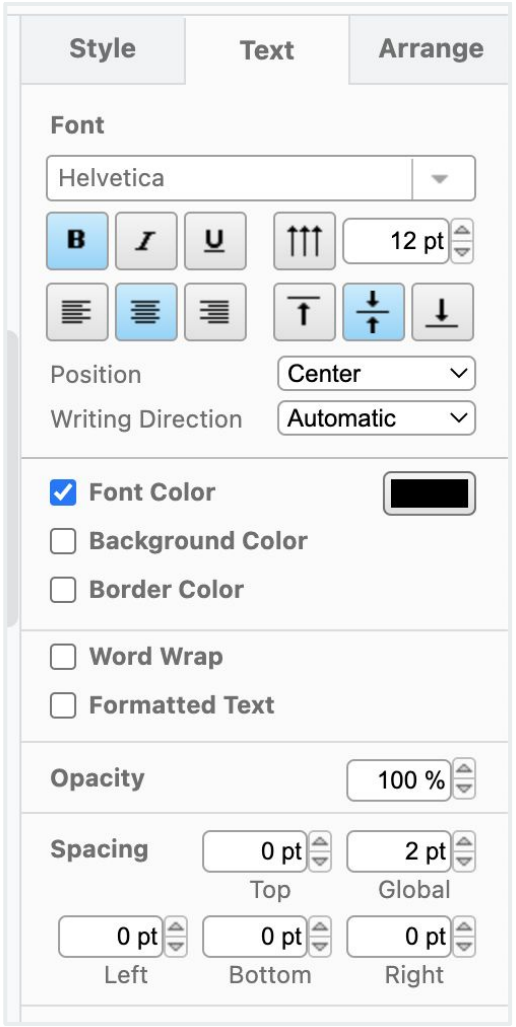


On the left panel, you will be able to find the shapes to create your diagrams. There are many different options, but the one we'll focus on is the ***Entity Relation*** section. At the top you'll find options to create entity structures and towards the button you'll see relationship lines (one-to-one, one-to-many, many-to-many) like we discussed earlier.

As you add shapes and different pieces on the page, you will get options to style on the right hand panel. You can provide different ***styling for the background and border*** and make changes to the ***style of the text*** as well. If you do not see these options, remember to select one of the shapes on the page.



[✓] Fill
[ ] Gradient
[ ] Lanecolor
[✓] Line 1 pt
Perimeter 0 pt
Opacity 100 %
[ ] Rounded
[✓] Divider
[ ] Shadow
[ ] Sketch



Font
Helvetica
12 pt
Position: Center
Writing Direction: Automatic
[✓] Font Color
[ ] Background Color
[ ] Border Color
[ ] Word Wrap
[ ] Formatted Text
Opacity 100 %
Spacing
0 pt Top
2 pt Global
0 pt Left
0 pt Bottom
0 pt Right

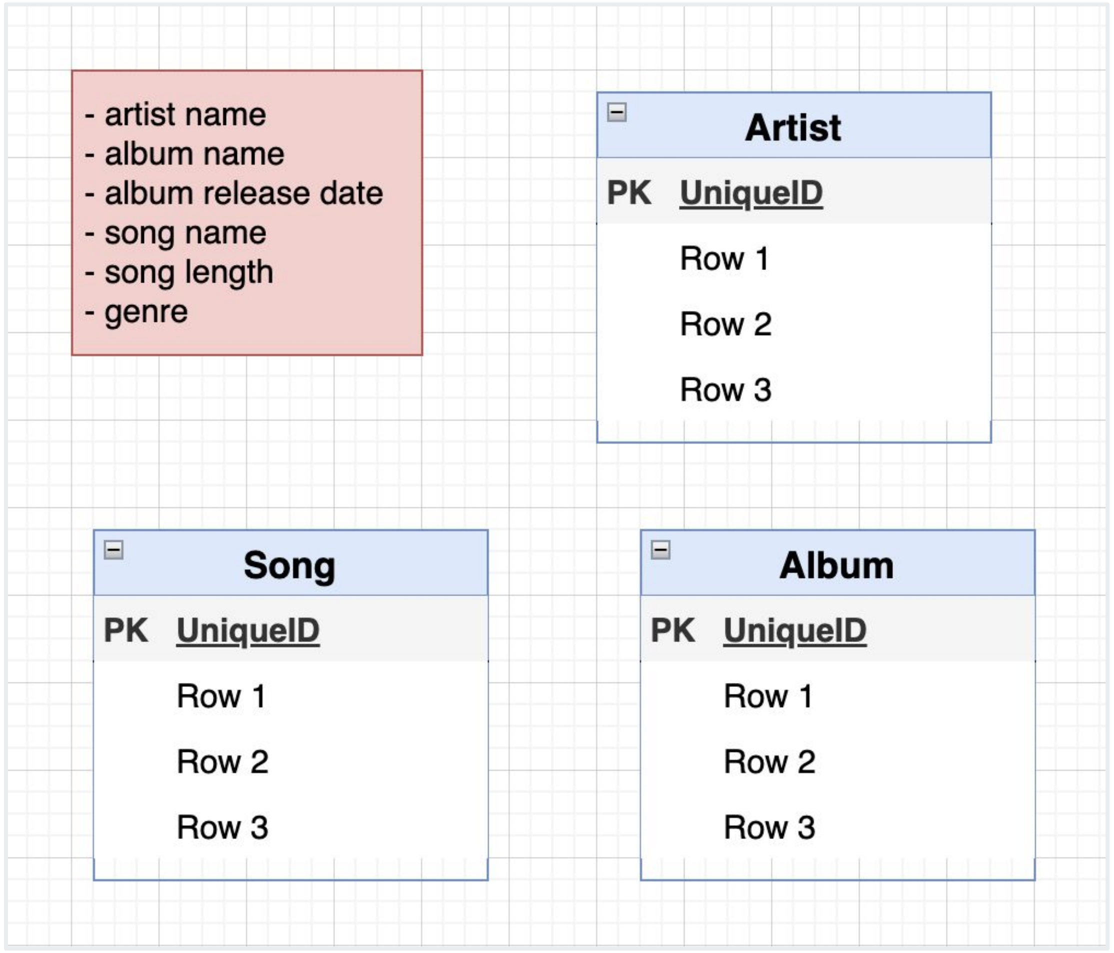
Find the Key Data

➔ First before starting to create the ER, we need to figure out all the pieces of data we need to store based on the requirements given

➔ We take our requirements of the left and **create a list of all the pieces of data that need to be stored** (i.e. what column data we need to have)

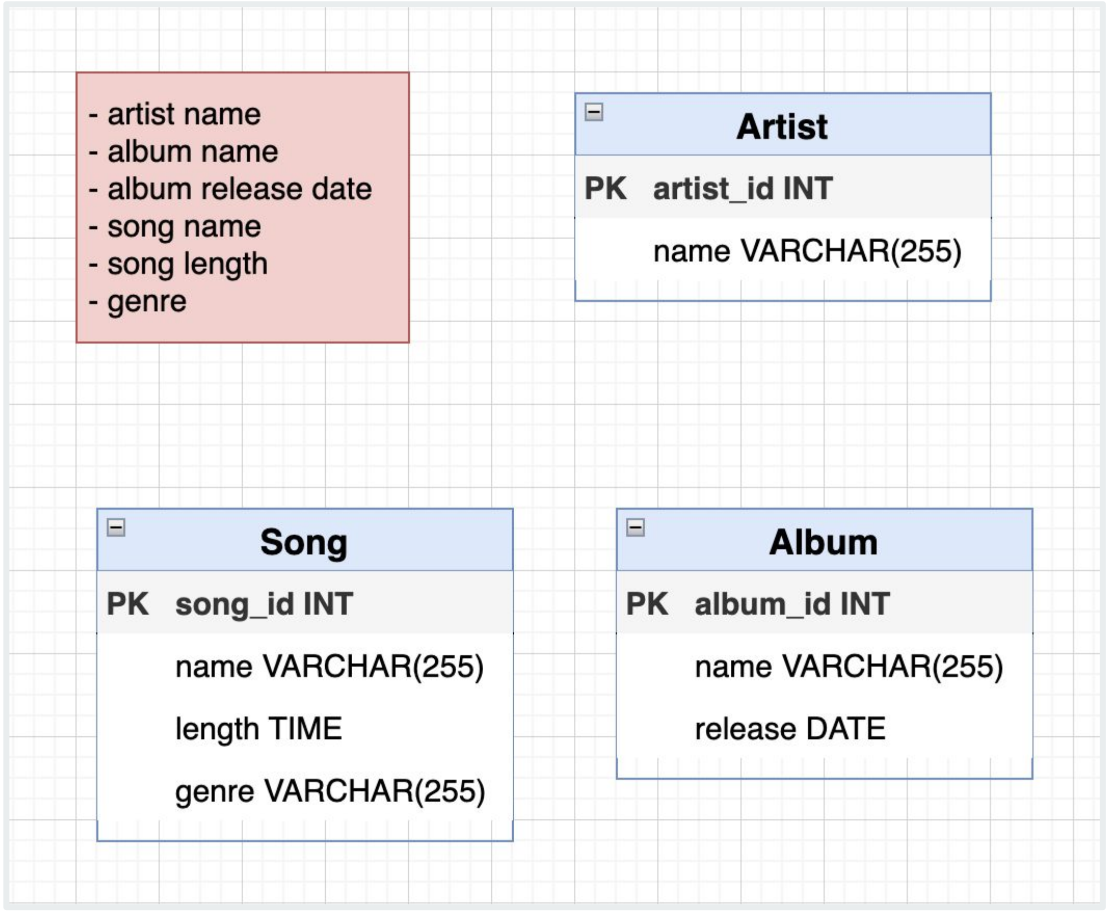
Scenario: You are building an application for a music streaming company. They need to store information for artists names, album names, album release date, genre, song name, song length. What kind of ER do you build to model your data?

- artist name
- album name
- album release date
- song name
- song length
- genre



Break Into Entities

- ➔ From our list, we can create our entities
- ➔ ***Do not worry about setting up any relationship tables***
- ➔ It is just important to **figure out from the data, what the main entities should be**



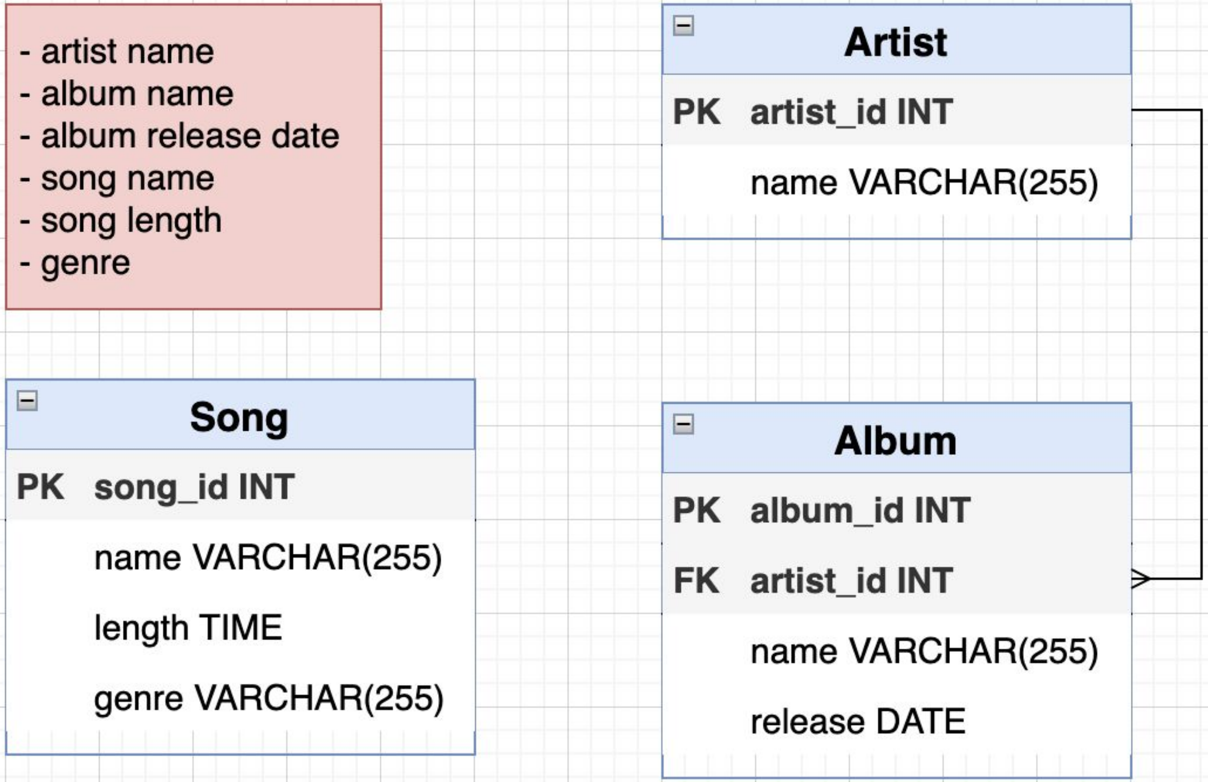
Split Into Columns

- ➔ Start taking the data we had listed before and now distribute it amongst the entities
- ➔ You ***can create id values*** if nothing listed makes sense for the primary keys

We can start off by creating a one-to-many relationship between artist and album. As we do so, we add in a foreign key.

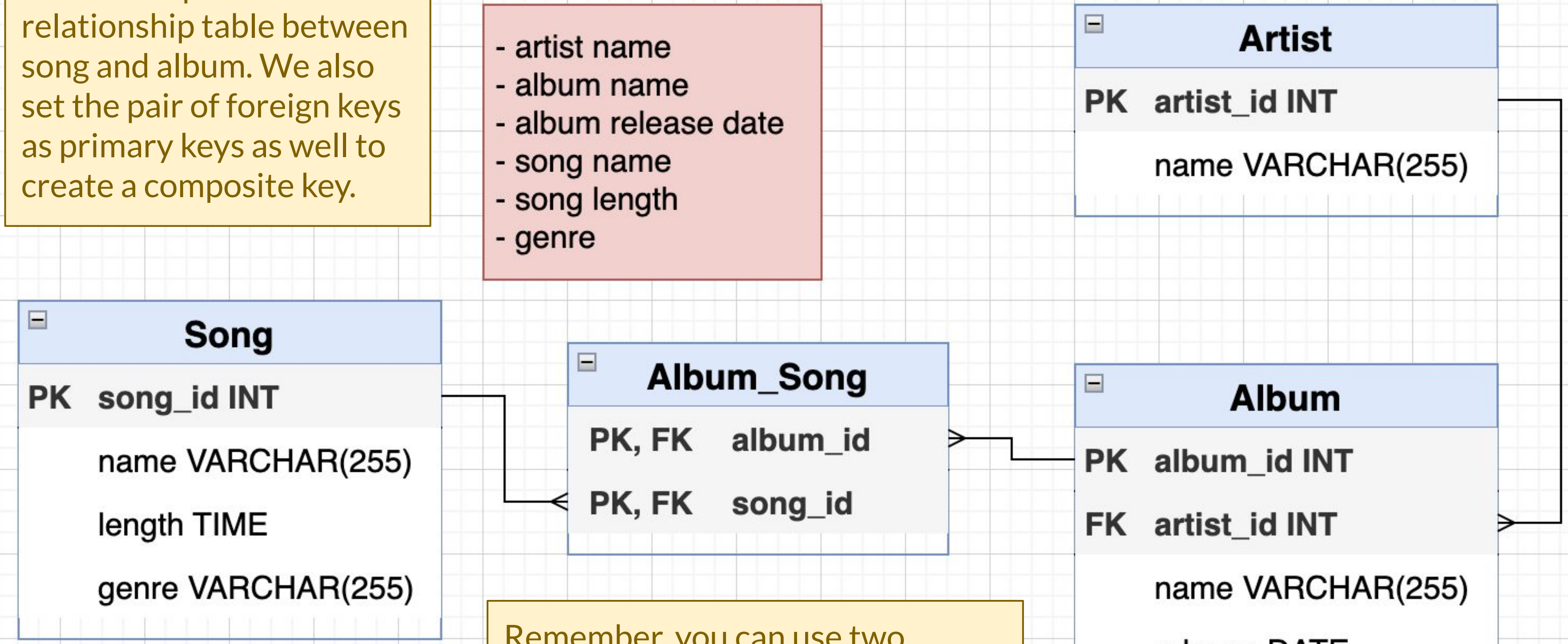
But how do we then connect Songs? We can say that it can have a one-to-many relationship as well to album. However, we do know that songs can be on multiple albums and that albums can have many songs.

- artist name
- album name
- album release date
- song name
- song length
- genre

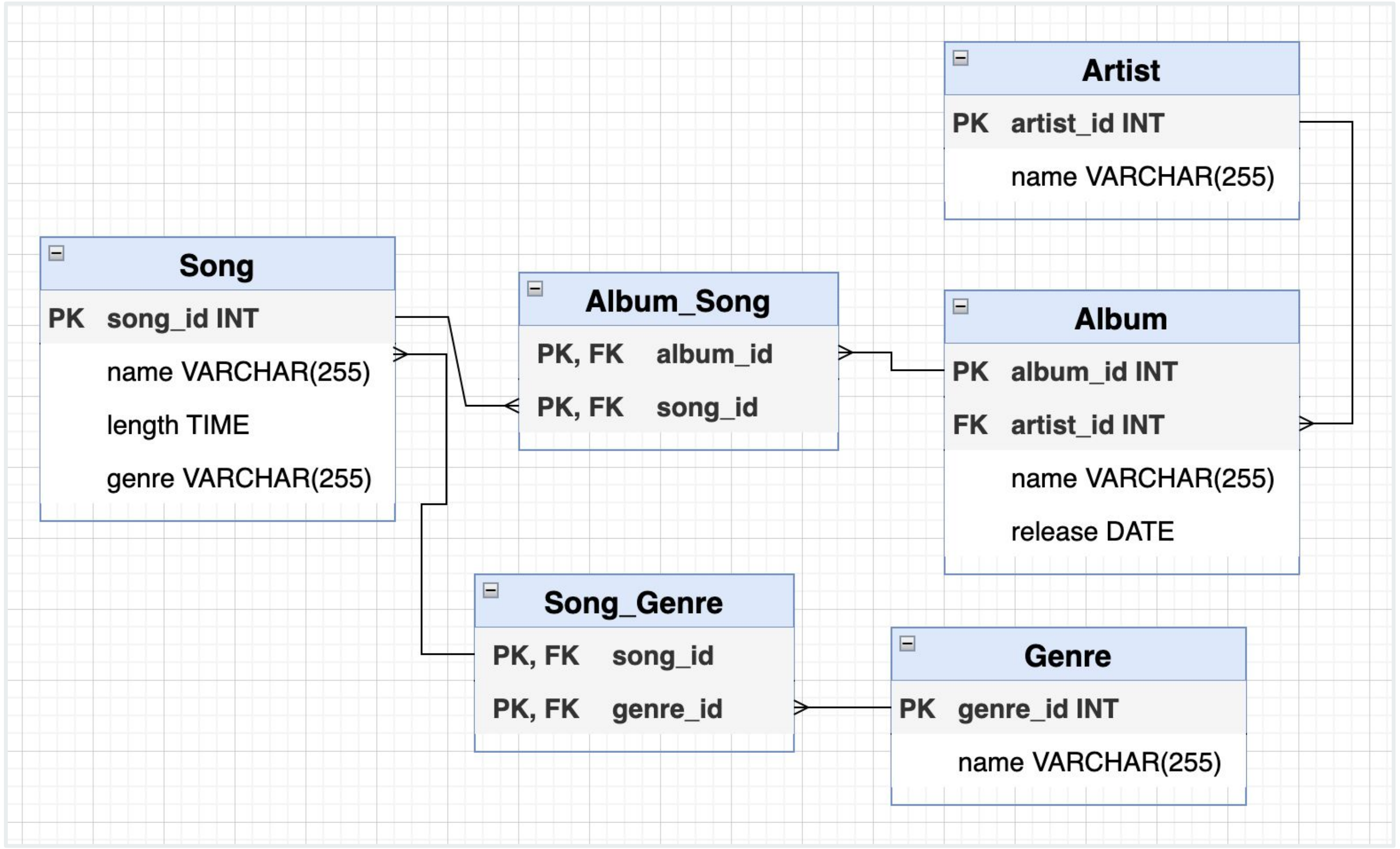


So we set up the relationship table between song and album. We also set the pair of foreign keys as primary keys as well to create a composite key.

- artist name
- album name
- album release date
- song name
- song length
- genre



Remember, you can use two one-to-many lines to set up the many-to-many with a relationship table.



Alternatively we can set the genre as a separate table and connect it to Song. Thus creating another many-to-many relationship on top of the other one present. To view this ER check this link: https://drive.google.com/file/d/1AJAsYyT3C9aruTaZb0ufO6eRxXG1DmYo/view?usp=sharing

Alternative Ideas

- ➔ We were able to modifier our previous ER to now include multiple genres to a song, but there are more ways to expand
- ➔ With the data we expected to store so far we haven't taken into account:
  - ◆ ***Can an album list multiple artists?***
  - ◆ ***How do we store information for feature artists on a song?***
  - ◆ ***Can albums have genres?***
- ➔ So even with the small sets of data we need to store, there can be many more relationships we can still describe

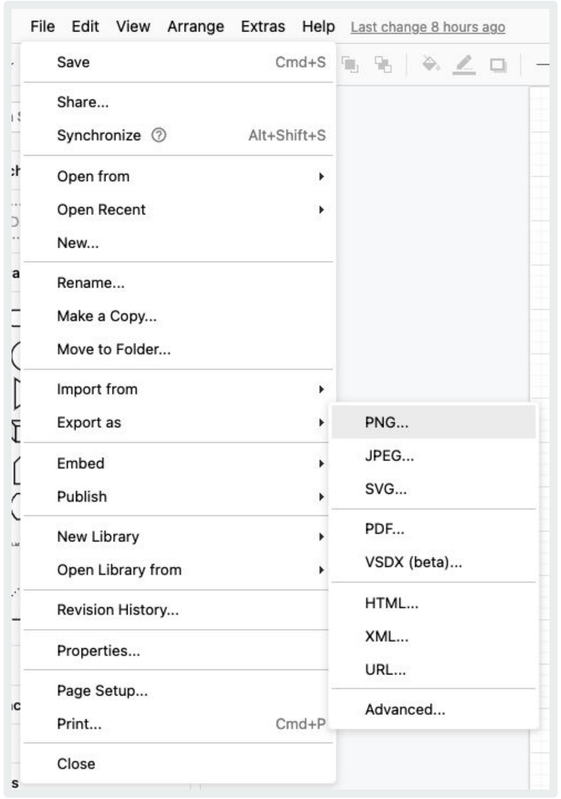
Homework #13: ER Exercise

Every time a new kid registers for the little league team for the new season, their information is ***stored in one big table*** (see below). The players **can have more than one guardian**, so at the moment, they need to insert another row into this table if they want to add the information of any other guardian. Keep in mind that the **coach for a team can change during different seasons** (they can retire). Normalize the data and display your answer as a ER diagram. The diagram does not need to be exact, it just need to show the ***table name, the attributes and their data types, and indicate any primary or foreign keys***. Feel free to split up the data into more columns or rename columns as needed.

Little League Registration				
player_id	player_number	guardian_email	coach_email	season_start_date
player_name	player_address	team_name	player_school	season_end_date
player_date_of_birth	guardian_name	coach_name	school_address	

Homework #13: Submission

- ➔ *Turn in a screenshot of the diagram*
- ➔ **Don't need to use Draw.io**, any diagram tool is acceptable
- ➔ If on ***Draw.io***, go to **File** → **Export as**, you will get several options on how you may want to download the diagram
  - ◆ Please select either ***PNG***, ***JPEG***, or ***PDF***
  - ◆ Other options will not be accepted
- ➔ Once screenshotted or downloaded, **please turn in your diagram through Github Classroom**
  - ◆ Upload the file directly to this repository
  - ◆ Name your diagram with a clear name like Little_League_Diagram or ER_Diagram



Open Book Quiz on ER, Wire Frames

- ➔ Check README.md
- ➔ This is an **open note**, multiple choice quiz
- ➔ Have it completed by the **start of class tomorrow at 10AM EST**
- ➔ If there are ***any questions, ask your instructor during this time or during office hours***, as they may not able available after hours

FIN