Lesson Goals

- Learn foundational database concepts such as entities and relationships.
- Learn about the different types of entity relationships and how to identify them.
- Learn about normalization and how entities and relationships are expressed in a database.
- Learn about entity relationship diagrams and how they represent a database's schema.

Introduction

In the prework for this program, we learned how to get started using MySQL and how to write queries with some of the basic SQL commands. Now that you know how to do that, it is important to take a step back and learn about the concept of relational databases before delving in to more advanced queries. This lesson will hopefully help you understand the "why" behind what you will learn later in this chapter.

Entities and Attributes

Data sets typically contain information about *entities*. You can think of entities as nouns - people, places, things, or even events. For example:

- In a data set about books, you are likely to find entities such as titles, authors, publishers, sales transactions, etc.
- In a data set about sports, you are likely to find entities such as players, positions, teams, games, seasons, plays, etc.
- In a retail data set, you are likely to find entities such as products, customers, stores, purchases, product categories, etc.
- In a movies data set, you are likely to find entities such as movies, roles, actors, directors, genres, etc.

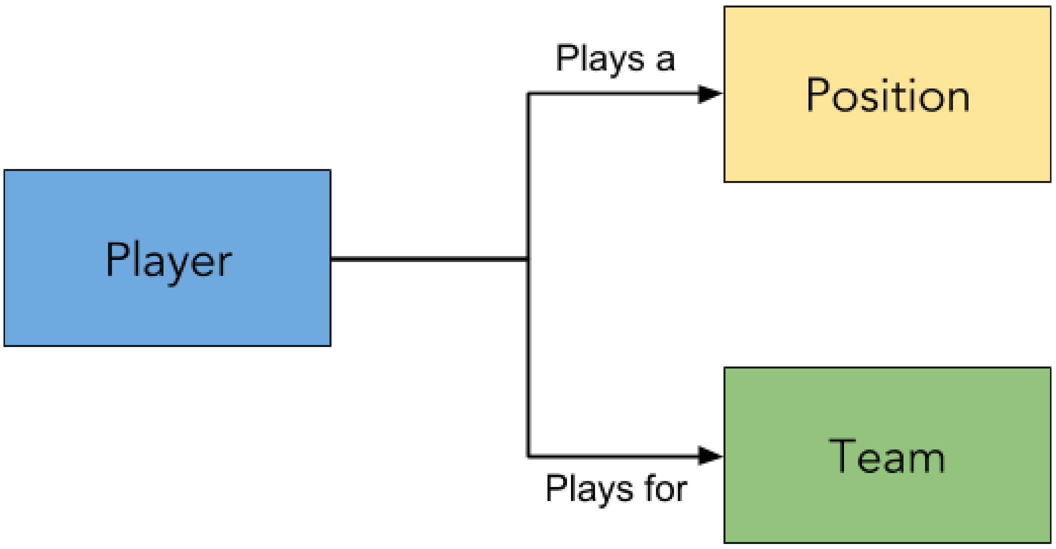
We need to get good at identifying entities in our data because they are ultimately going to be the subjects we are analyzing.

In a relational database, entities are represented by rows in a table. These rows typically span across multiple columns, which represent *attributes* that contain information about each entity. In the table below, the rows represent stores (entities) and for each store, we have a unique *store_id* field as well as *stor_name*, *stor_address*, *city*, *state* and *zip* columns (attributes).

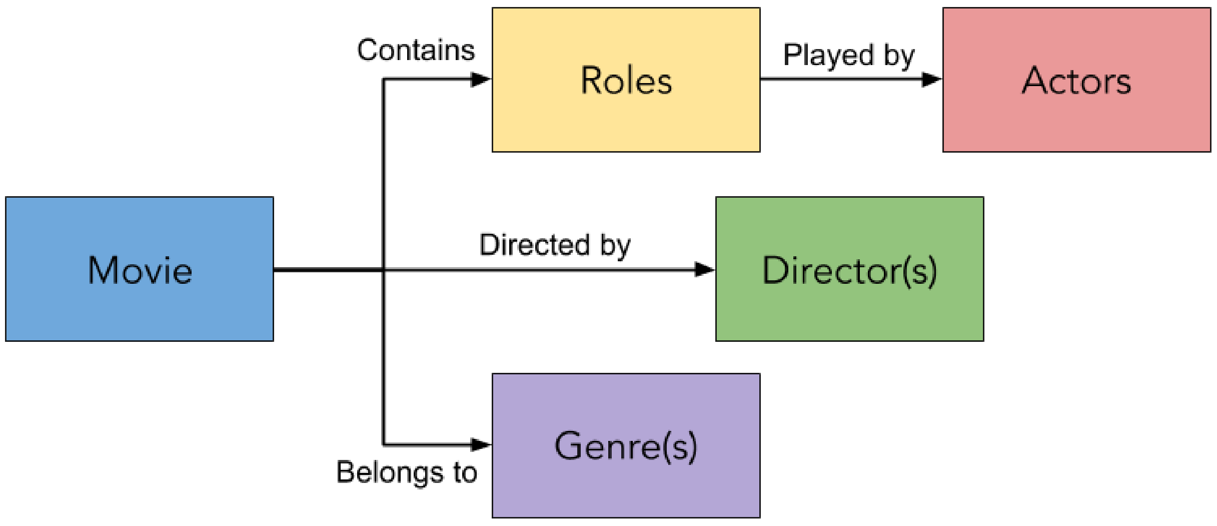
stor_id	stor_name	stor_address	city	state	zip
6380	Eric the Read Books	788 Catamaugus Ave.	Seattle	WA	98056
7066	Barnum's	567 Pasadena Ave.	Tustin	CA	92789
7067	News & Brews	577 First St.	Los Gatos	CA	96745
7131	Doc-U-Mat: Quality Laundry and Books	24-A Avogadro Way	Remulade	WA	98014
7896	Fricative Bookshop	89 Madison St.	Fremont	CA	90019
8042	Bookbeat	679 Carson St.	Portland	OR	89076

Relationships

The entities in our data also can be related to other entities. For example, in the case of a sports data set, a player can play one or more positions for a specific team. In other words, there exists a relationship between the player and the position and also between the player and the team.



These relationships can range from very simple to more complex. For example, in a movie data set, a particular movie is going to consist of many roles (each played by an actor), have one or more directors, and belong to one or more genres.



In a relational database, these are structured as one of three types of relationships.

One-to-One

A one-to-one relationship exists when a specific entity can have, or belong to, only one other specific entity *and vice versa*. For example, each U.S. citizen has only one Social Security number, and a specific Social Security number can belong to only one person. True one-to-one relationships are actually the most difficult to find in the real world, and many people confuse them with the next type of relationship.

One-to-Many

A one-to-many relationship exists when an entity can have, or belong to, only one other specific entity, but the same is not true of the other entity. For example, a player can play for only one team at a time but a team consists of multiple players. This depends, however, on the scenario upon which your data is modeled. If you are looking at a specific season, then the relationship holds, but from one season to another, a player may decide to switch teams, in which case the relationship would be identified as the next type we are going to discuss.

Many-to-Many

In a many-to-many relationship, one specific entity can have, or belong to, many other entities and vice versa. For example, a movie can have multiple actors in it, and actors can star in many different movies. Similarly, a book may have multiple authors, and an author may have written multiple books.

It is important to think about the different entities and relationships that exist in your data so that you can design your database appropriately. Having an understanding of these will also inform the types of analysis you can perform. In the next couple sections, we will look at how to leverage these concepts in the design of a database.

Data Normalization

The data in a relational database should be organized in a *normalized* fashion. This means that different entities in the data will have their own respective tables where each individual entity will be listed once with a unique ID assigned to it. This unique ID is called a *primary key*.

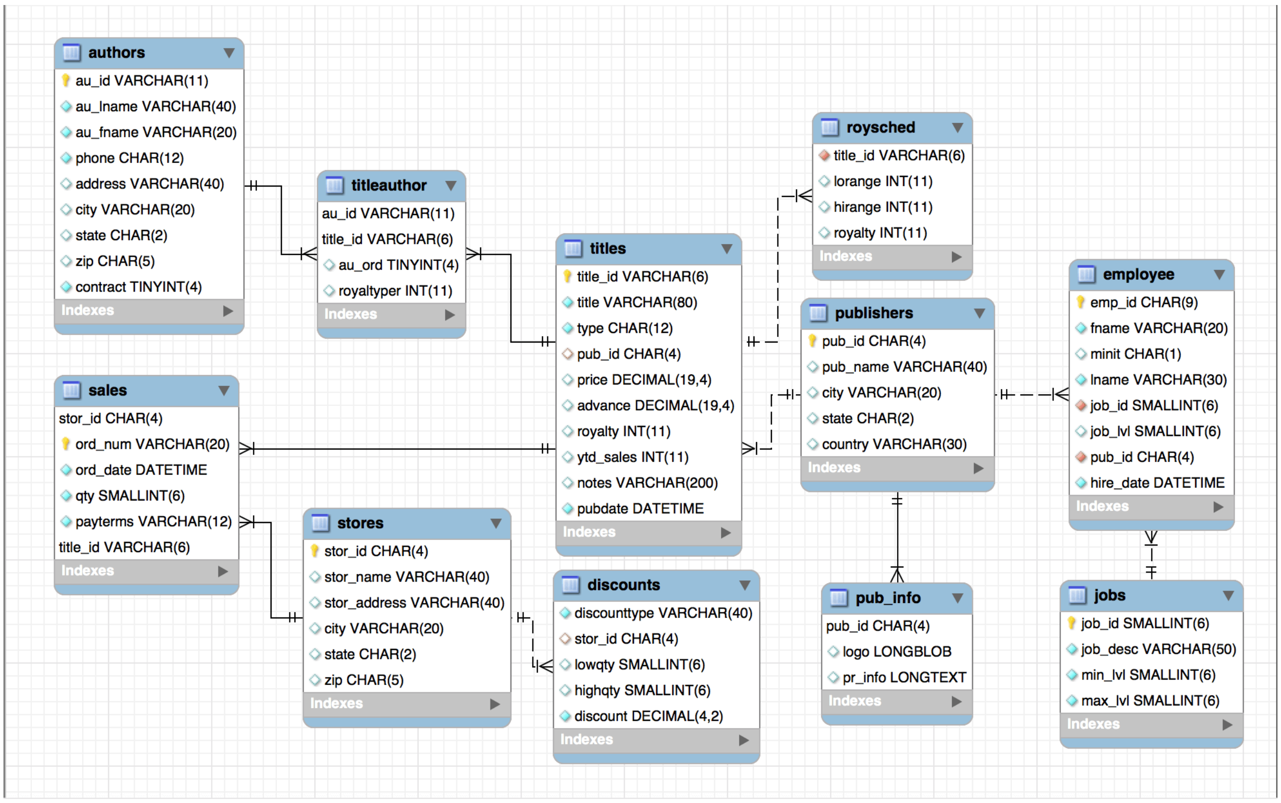
The relationships between entities will be managed by relationships inside the database where the ID for one entity will be mapped to the ID for the related entity by a *foreign key*. A foreign key is a field in one table that refers to an identifier for an entity in another table. Queries are built on top of these entity tables and leverage these relationships to show specifically the information that the user would like to see.

One of the main advantages of storing data this way is to avoid redundancy. For example, if a player on Team A moved to Team B, you would only need to update the relationship for that player in the table that contained player-team relationships, and it would automatically get updated in every query that includes what team a player plays for.

Relational Database Design

Now that we have covered the main aspects of a relational database, let's learn how to design one. If you wanted to design a normalized database, you would need to take stock of the entities and relationships you have in your data, split the entities up into their own tables, and then create the relationships in the database's schema.

An *entity relationship diagram* (ERD) is a visual representation of a that schema. It allows you to get an overview of what tables are in your database and how those tables relate to each other. Below is an example ERD for a publications database.



As you can see, the diagram includes tables for each type of entity, and the lines joining the tables together represent the different entity relationships that are present in the schema of this database. For example, we see a one-to-many relationship between the `titles` table and the `sales` table and also between the `publishers` table and the `employees` table. There are also a couple many-to-many relationships in this database, such as the those between the `titles` and `authors` tables and between the `stores` and `titles` tables. Note that these many-to-many relationships are established via intermediary relationship tables (`titleauthor` and `sales` respectively) that are both part of one-to-many relationships on both sides.

Also, notice that the primary keys are labeled with a little key icon next to their field names. The foreign keys are a little more difficult to spot, but they include the *au_id*, *title_id*, *store_id*, and *pub_id* fields referenced in tables other than those in which they are the primary key.

Summary

In this lesson, we took a step back from SQL to learn about foundational concepts in relational database design. First, we covered the concepts of entities and relationships in data. Then, we learned about the different types of relationships that can exist and looked at a few examples to help us identify them. We then discussed how these concepts are actually represented in a relational database via normalization. Finally, we introduced entity relationship diagrams and looked at an example so that you could get a visual sense of the different components of a database. Now that we have an understanding of these basic concepts, we are ready to dive back into SQL and write some queries that leverage what we have learned here.