Second Normal Form

by Sophia

WHAT'S COVERED

This tutorial explores the second normal form (2NF) built on a first normal form (1NF) design in two parts:

1. Introduction
2. Moving Ratings Example

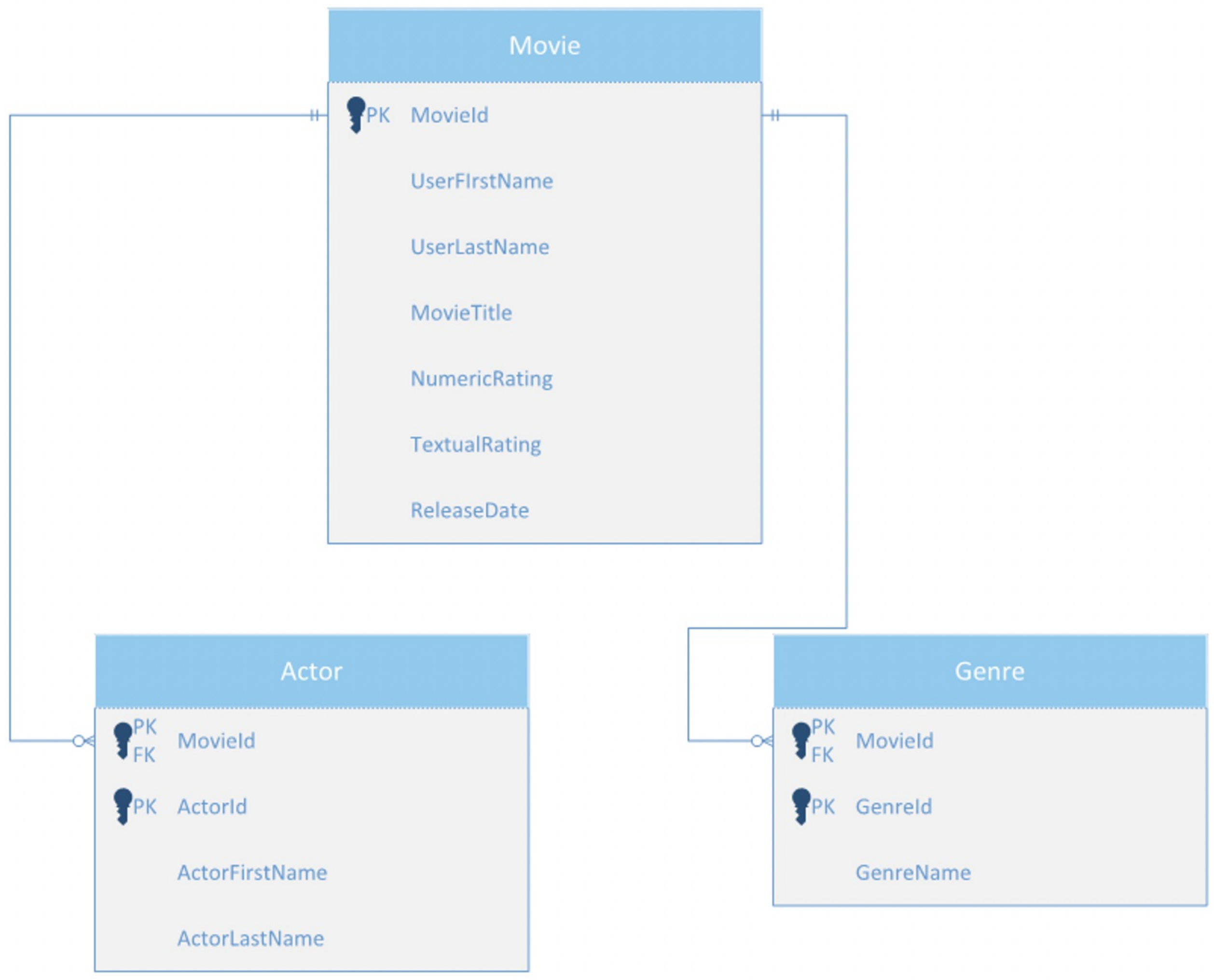
1. Introduction

With the second normal form (2NF), the database has to, first of all, be in the first normal form (1NF). Once it is in the first normal form (1NF), the other criterion is to ensure that each non-key attribute is functionally dependent on the primary key. As a reminder, functional dependency means that each column in a table that is not a primary key should be determined by that primary key. You will need to review each column to determine if it is dependent on and specific to the primary key in each table.

2. Moving Ratings Example

For the movie ratings database, at the end of the first normal form (1NF) you had the following tables:

- Movie (MovieId, UserFirstName, UserLastName, MovieTitle, NumericRating, TextualRating, ReleaseDate)
- Actor (MovieId, ActorId, ActorFirstName, ActorLastName)
- Genre (MovieId, GenreId, GenreName)



Let's now review each column in the Movie table to determine if it is dependent on and specific to the primary key:

- MovieId: The MovieId is the primary key of the Movie table, so you can ignore it.
- UserFirstName: No, the UserFirstName does not depend on the MovieId, as it is specific to the user that is creating the rating.
- UserLastName: No, the UserLastName does not depend on the MovieId, as it is also specific to the user that is creating the rating.
- MovieTitle: Yes, the MovieTitle should be directly dependent on the MovieID, as a different MovieId should also mean a different MovieTitle.
- NumericRating: No, the NumericRating is not unique to the MovieId. A MovieId could have many different NumericRatings by different users.
- TextualRating: No, the TextualRating is not unique to the MovieId. A MovieId could have many different TextualRating by different users.
- ReleaseDate: Yes, the ReleaseDate should be directly dependent on the MovieID, as a different MovieId should also mean a different ReleaseDate.

In the Actor table:

- MovieId, ActorId: These are primary keys, so for now, you can ignore them.
- ActorFirstName: Not entirely. The ActorFirstName is dependent on the ActorId, but not on the MovieId. This creates an issue with a many-to-many relationship between the Movie and Actor table.

- ActorLastName: Not entirely. The ActorLastName is dependent on the ActorId, but not on the MovieId. This also creates an issue with a many-to-many relationship between the Movie and Actor table.

In the Genre table:

- MovieId, GenreId: These are primary keys, so for now, you can ignore them.
- GenreName: Not entirely. The GenreName is dependent on the GenreId, but not on the MovieId. This creates an issue with a many-to-many relationship between the Movie and Genre table.

Let's begin to resolve these issues.
Starting with the UserFirstname and the UserLastName in the Movie table. Both of these are unique to a user, although two different users could have the same name. This could create some problems in our data, so you need to create a User table with a primary key which you will call UserId:

- User (UserId, UserFirstName, UserLastName)

Next, you need a Rating table. The rating is related to the User and the Movie, and should contain the UserId and MovieId as foreign keys, along with the NumericRating and TextualRating:

- Rating (UserId, MovieId, NumericRating, TextualRating)

Then, you need to determine if those two foreign keys combined would be unique. This depends on the business rules, and whether a user can submit more than one rating per movie. If a user cannot post more than one rating per movie, the combined values of the UserId and MovieId can be set as a composite primary key. However, since this was not specified in our business rules, it would be safer to create a separate primary key to allow the possibility of a user being able to add more than one rating for a movie. Adding a RatingID as a primary key, the Rating table now looks like this:

- Rating (RatingId, UserId, MovieId, NumericRating, TextualRating)

Our Movie table now contains only the following:

- Movie (MovieId, MovieTitle, ReleaseDate)

Next, you turn our attention to the Actor and Genre tables, which have columns that do not fully depend on the primary key due to the many-to-many relationships that are still in place.

- Actor (MovieId, ActorId, ActorFirstName, ActorLastName)
- Genre (MovieId, GenreId, GenreName)

As you've established, an actor can act in many movies, and a movie can have many actors. However, an actor in a movie acts in a specific role. So our many-to-many relationship can be defined by a bridge table named Role to join the two tables of Actor and Movie together:

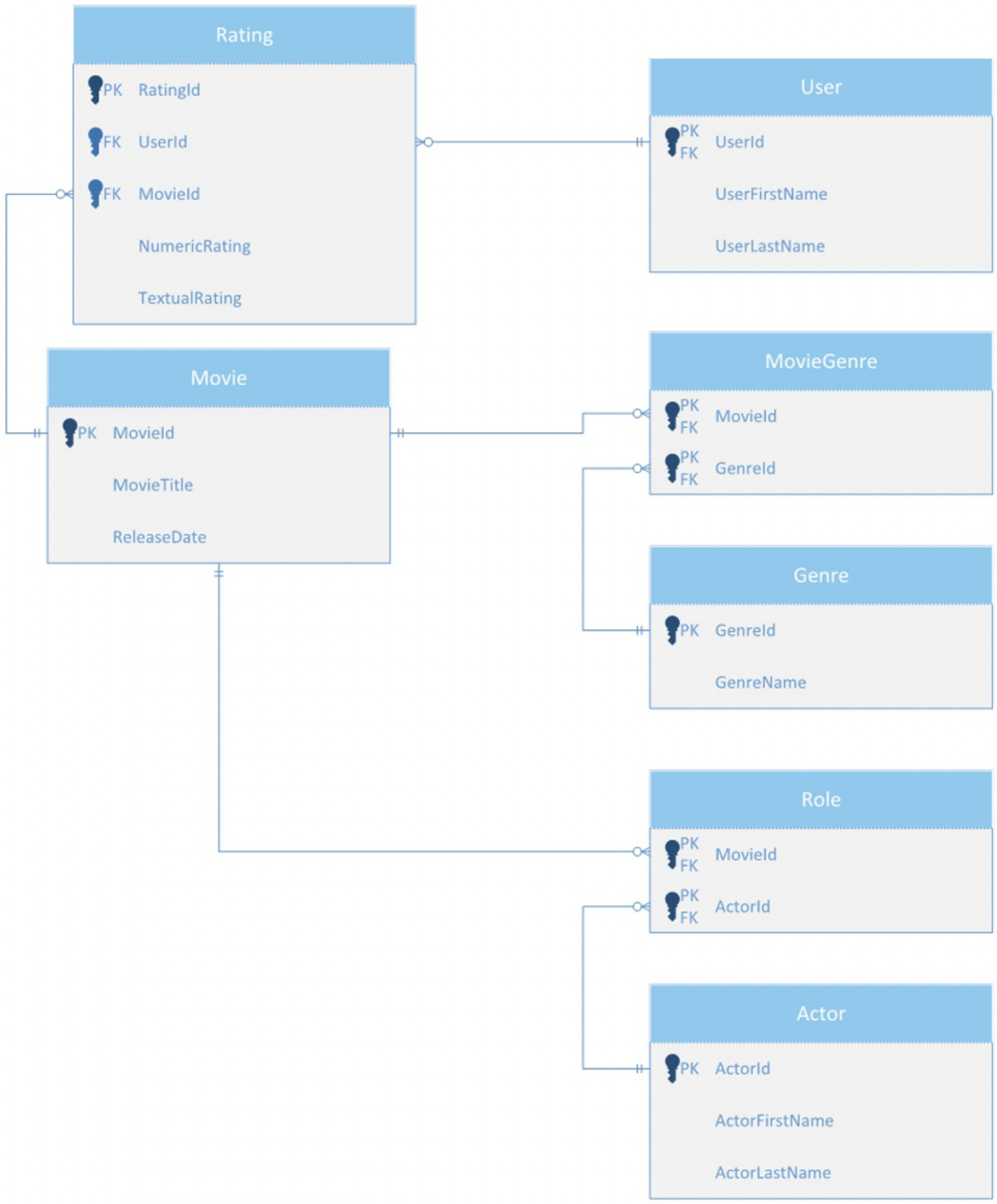
- Movie (MovieId, MovieTitle, ReleaseDate)
- Role (ActorId, MovieId)
- Actor (ActorId, ActorFirstName, ActorLastName)

The ActorId and MovieId can be set up as a composite primary key.
You can also build a bridge table for the many-to-many relationship between Movie and Genre, called

MovieGenre:

- Movie (MovieId, MovieTitle, ReleaseDate)
- MovieGenre (MovieId, GenreId)
- Genre (GenreId, GenreName)

The GenreId and MovieId can be set up as a composite primary key in this bridge table. The resulting ERD for our second normal form (2NF) would look like the following:



SUMMARY

In this tutorial, you learned in the **introduction** that first and foremost, before a database can enter the second normal form (2NF) stage, it needs to be in the first normal form (1NF) first. Then you learned how to ensure dependency on the primary key using the **movie rating example**. Additional tables were created for the completion of the final second normal form (2NF).

Source: Authored by Vincent Tran