Vertabelo

← BACK TO ARTICLES LIST

December 3, 2020 - 10 minutes read

TECHNICAL

What Is a Many-to-Many Relationship in a Database? An Explanation with Three Examples

Tihomir Babic
Database designer

Tags: data modeling | database design | many-to-many relationship

What is a many-to-many relationship in database modeling? How do you implement this relationship in a database? The examples in this article will answer these questions.

how to implement many-to-many relationships in a relational database.

Ready? Let’s get started.

Many-to-Many Relationship in Theory

A many-to-many (or M:N) relationship is one of the three database relationships. The other two are:

- One-to-one (1:1) relationships
- One-to-many (1:N) relationships

By definition, a many-to-many relationship is where more than one record in a table is related to more than one record in another table. Such a relationship can be tricky to represent in the database, so I’ll show you how to do it in the following example. You might also want to READ ABOUT ENTITIES, ATTRIBUTES, AND HOW TO DEFINE THEM. I’ll use those concepts a lot, so reading the article won’t hurt.

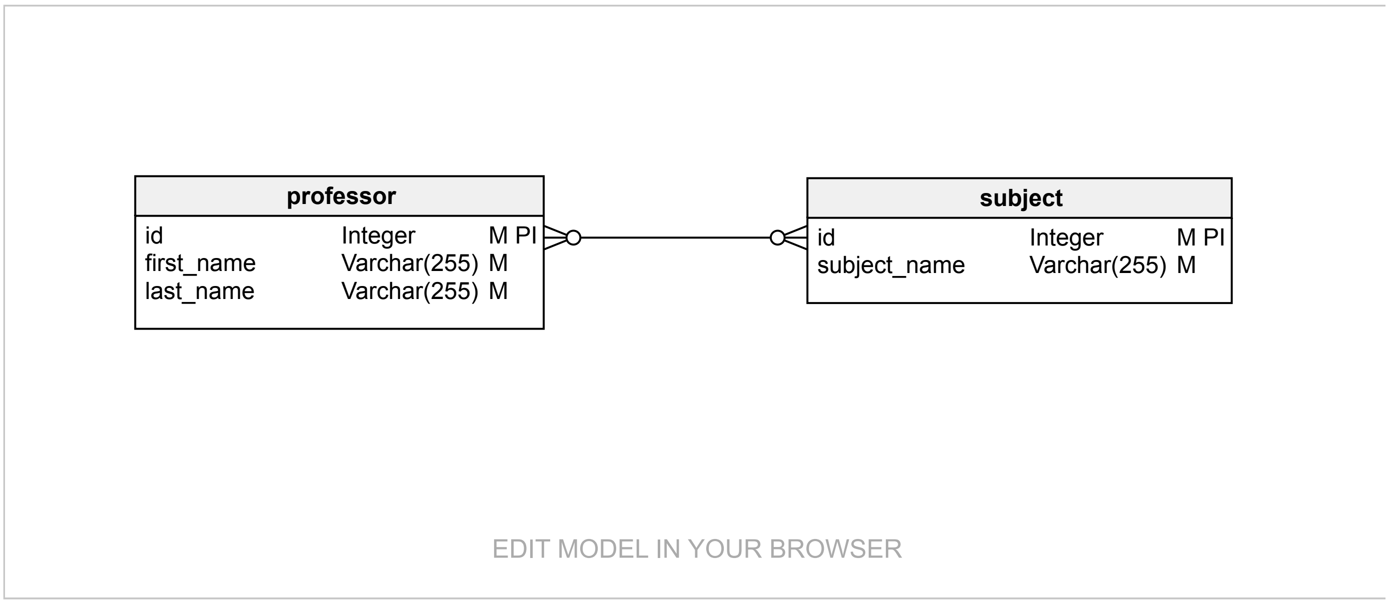
Many-to-Many Relationship in Practice

Let’s move on to the three examples. I’ll show these many-to-many relationships in ER diagrams drawn in the VERTABELO DATABASE MODELER.

Example 1: University Database

In this example, your task is to build a university database. You’ve just started, but you’re already stuck. You have to show the professors and the subjects they teach, but how will you do it?

relationship, so here's how your logical model should look:


professor
id Integer M PI
first_name Varchar(255) M
last_name Varchar(255) M
subject
id Integer M PI
subject_name Varchar(255) M
EDIT MODEL IN YOUR BROWSER

The `professor` entity has the following attributes:

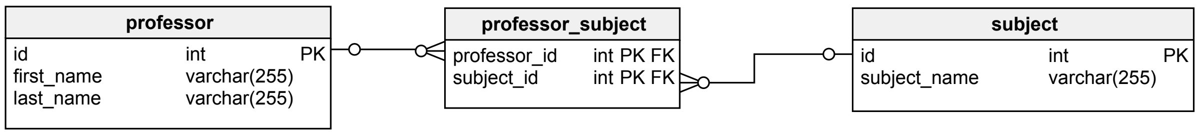
- `id`: The professor's ID; a primary identifier (PI).
- `first_name`: The professor's first name.
- `last_name`: The professor's last name.

The `subject` entity has the attributes:

- `id`: The subject's ID; a primary identifier (PI).
- `subject_name`: The subject name.

Many-to-many relationships are not ideal. If left as it is in the above example, the data would be duplicated. For instance, if there's a professor that teaches six subjects, you would have him or her listed in the table six times, every time for a different subject. This is quite inefficient. So, how would you resolve this many-to-many relationship between these two entities? By introducing a junction table into your model. It will resolve the many-to-many relationship into multiple one-to-many relationships.

Vertabelo



EDIT MODEL IN YOUR BROWSER

As you can see, there's a new table called `professor_subject`. It contains the following attributes:

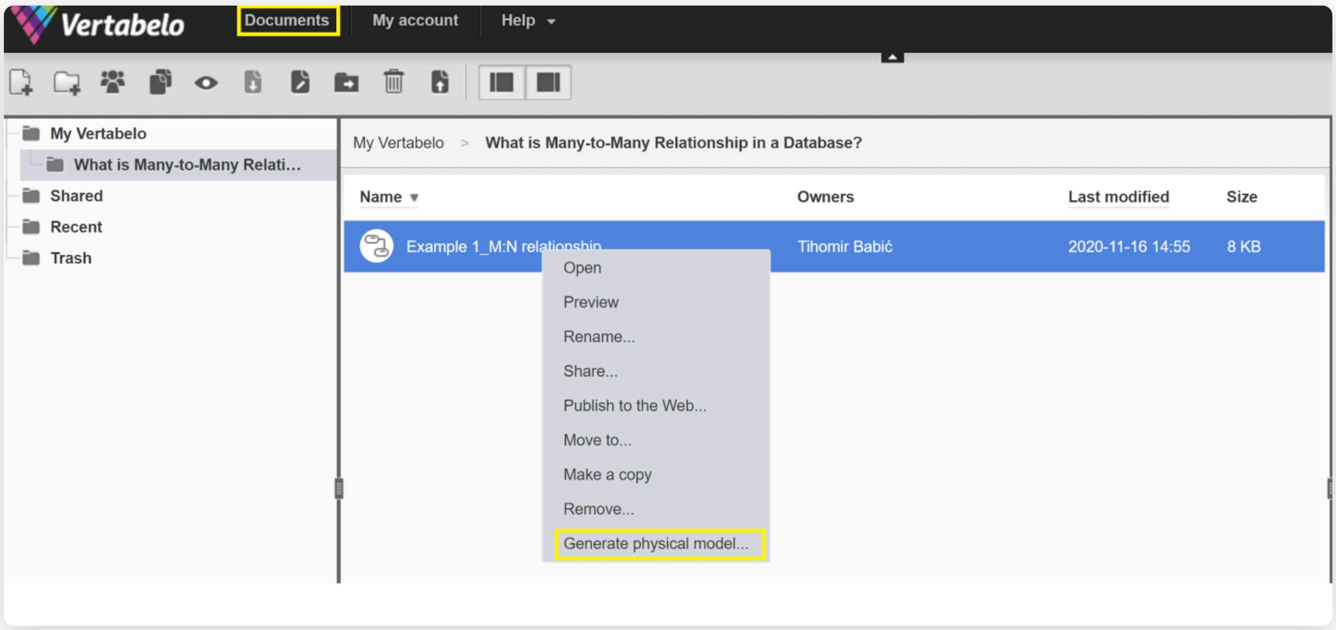
- `professor_id`: The professor's ID, which references the `id` column in `professor`.
- `subject_id`: The subject's ID, which references the `id` column in `subject`.

The column `professor_id` is a foreign key to the table `professor`. The same goes with the `subject_id`; it's a foreign key to the table `subject`. At the same time, the pair `professor_id`, `subject_id` is the primary key for the table `professor_subject`.

The columns `professor_id` and `subject_id` together form a composite primary key (i.e. the primary key consists of two or more columns). This composite primary key ensures that a professor can be assigned to one subject one time. Each pair of values (`professor_id`, `subject_id`) can be in the table no more than once. The same goes for the subjects; each one can be assigned to one professor one time. The composite key ensures the uniqueness of the attribute combinations.

Let's check that the junction table solves the many-to-many relationship. One professor can be allocated only once to the same subject. On the other hand, one again. There is a nice little trick provided by the Vertabelo Database Modeler that transforms a logical model into a physical one in four clicks. Here's how to do it:

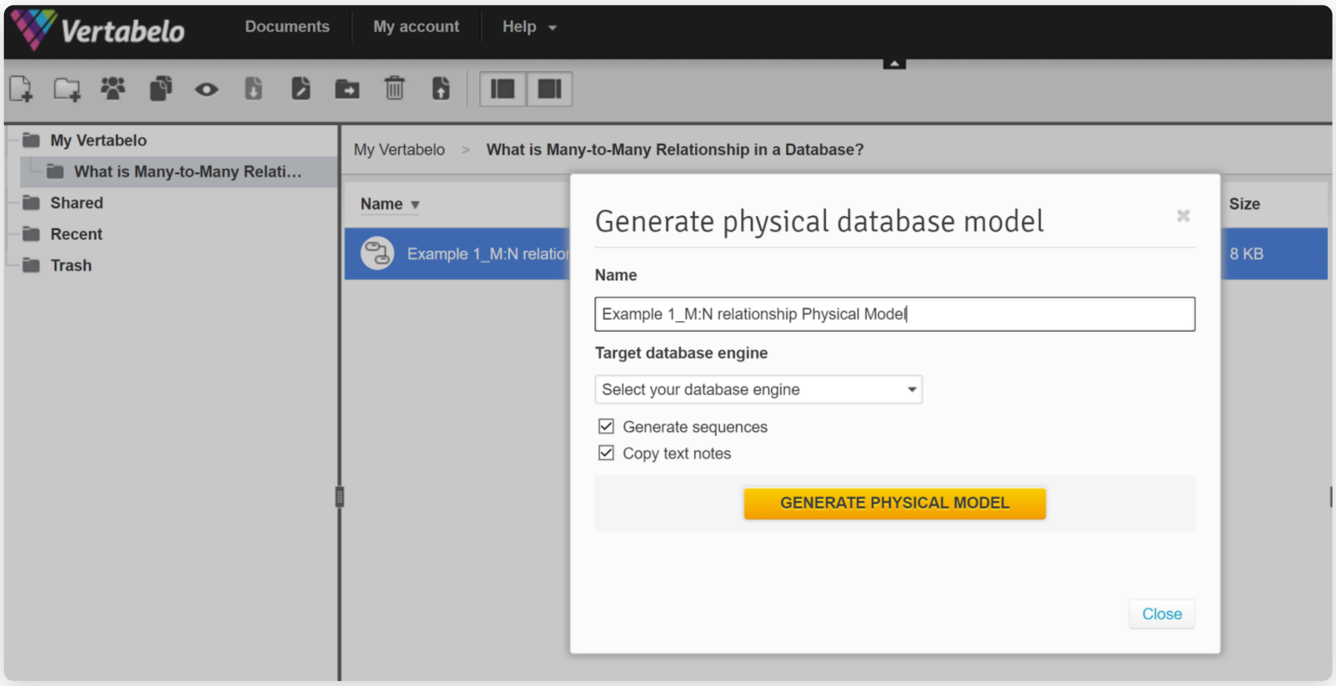
1. In your *Documents*, right-click on the model you want to make into a physical model. Choose the option "Generate physical model…"



My Vertabelo > What is Many-to-Many Relationship in a Database?

Name	Owners	Last modified	Size
Example 1_M:N relationship	Tihomir Babić	2020-11-16 14:55	8 KB

2. Then you'll see the dialogue box where you have to choose your target database engine. Select it (that's your third click), and click "Generate Physical Model".



My Vertabelo > What is Many-to-Many Relationship in a Database?

Generate physical database model

Name: Example 1_M:N relationship Physical Model

Target database engine: Select your database engine

☑ Generate sequences
☑ Copy text notes

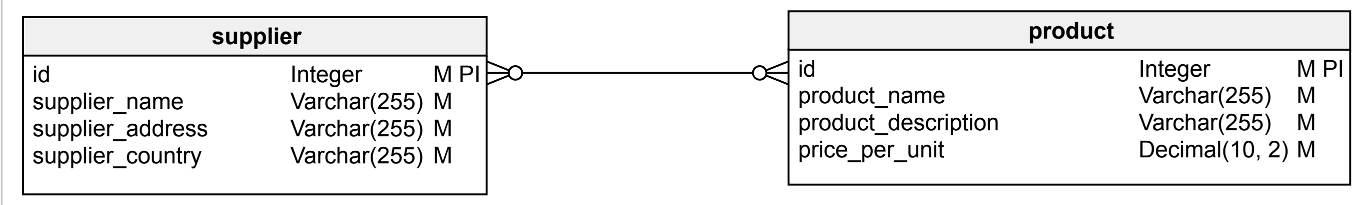
Vertabelo

EASIER.

Let’s move to the next example. It’s a little more complex, but don’t be afraid!

Example 2: Product Ordering Database

In this example, your task is to create a database that will help a company store information about their suppliers. The database will also contain info on all the products/services ordered from the suppliers. The logical data model could look something like this:


supplier
id Integer M PI
supplier_name Varchar(255) M
supplier_address Varchar(255) M
supplier_country Varchar(255) M
product
id Integer M PI
product_name Varchar(255) M
product_description Varchar(255) M
price_per_unit Decimal(10, 2) M

EDIT MODEL IN YOUR BROWSER

Once again, we have two entities:

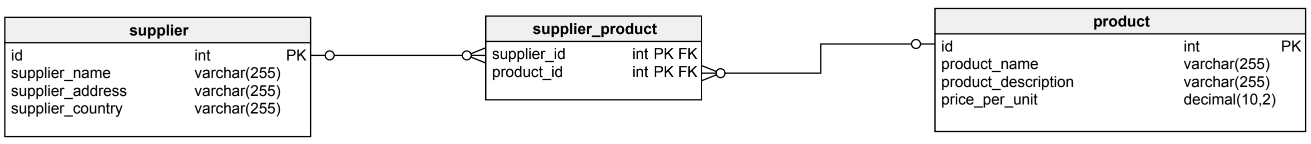
- `supplier`
- `product`

In the `supplier` entity, there are four attributes:

- `id`: The supplier’s ID; a primary identifier (PI).
- `supplier_name`: The supplier’s name.
- `supplier_address`: The supplier’s address.

- `product_name`: The product's name.
- `product_description`: The product's description.
- `price_per_unit`: The product's price per unit.

The relationship between those two entities is again many-to-many. One or many products can be ordered from one supplier. At the same time, the company can order the same product from many suppliers, e.g. services from different legal firms, tires from different manufacturers, etc. How would this logical model look when transformed into a relational database model? Like this:

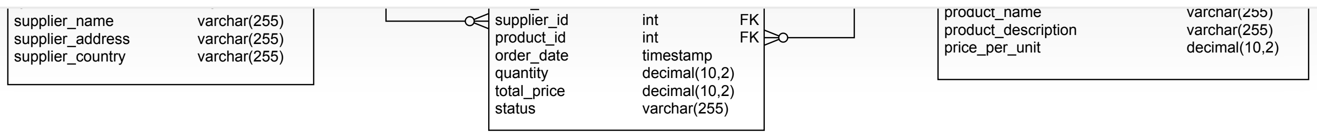

supplier
id int PK
supplier_name varchar(255)
supplier_address varchar(255)
supplier_country varchar(255)
supplier_product
supplier_id int PK FK
product_id int PK FK
product
id int PK
product_name varchar(255)
product_description varchar(255)
price_per_unit decimal(10,2)

EDIT MODEL IN YOUR BROWSER

Once again, instead of many-to-many, there's a new table that's automatically been named `supplier_product`. It has only two attributes:

- `supplier_id`: References the id column in the `supplier` table.
- `product_id`: References the id column in the table `product`.

Again, the pair `supplier_id`, `product_id` is the primary key of the table. However, this might not be enough! If the task is to create a database that will record orders from suppliers, it would be better to expand the table `supplier_product` a bit, as I've done below:


supplier_name varchar(255)
supplier_address varchar(255)
supplier_country varchar(255)
supplier_id int FK
product_id int FK
order_date timestamp
quantity decimal(10,2)
total_price decimal(10,2)
status varchar(255)
product_name varchar(255)
product_description varchar(255)
price_per_unit decimal(10,2)

EDIT MODEL IN YOUR BROWSER

It looks much better now! First of all, I changed the table's name to something more descriptive; it's now named `order`. I've also added several new attributes to the table. It consists of the following:

- `order_id`: The ID of this order from the supplier and the table's primary key (PK).
- `supplier_id`: The ID of the supplier; references the table `supplier`.
- `product_id`: The ID of the product ordered; references the table `product`.
- `order_date`: The date of the order.
- `quantity`: The number of items ordered.
- `total_price`: The total value of the ordered products.
- `status`: The status of the order.

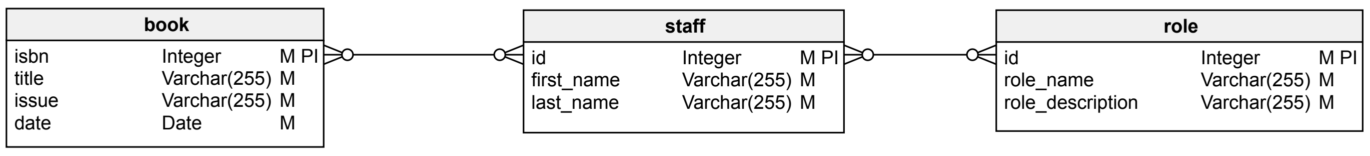
Remember that there are two possibilities when creating a junction table. One is that it contains only foreign keys that reference other tables, which happens often enough. However, sometimes the junction table becomes its own entity, as in this example where it also contains other attributes. You should always adapt the model to your needs.

If you want to practice a bit more on a similar example, see **THIS ARTICLE**.

Now that you've become so good at this, let's take a look at one more example!

Example 3: Book Publisher Database

Vertabelo



EDIT MODEL IN YOUR BROWSER

This time, there are three entities:

- `book`
- `staff`
- `role`

The `book` entity contains these attributes:

- `isbn`: The International Standard Book Number, a primary identifier (PI) used for books.
- `title`: The title of the book.
- `issue`: The issue (i.e. edition) of the book (e.g. first printing, first edition, etc.).
- `date`: The date of the issue.

The next entity is `staff`:

- `id:` The staff member's unique ID; a primary identifier (PI).
- `first_name`: The staff member's first name.
- `last_name`: The staff member's last name.
-

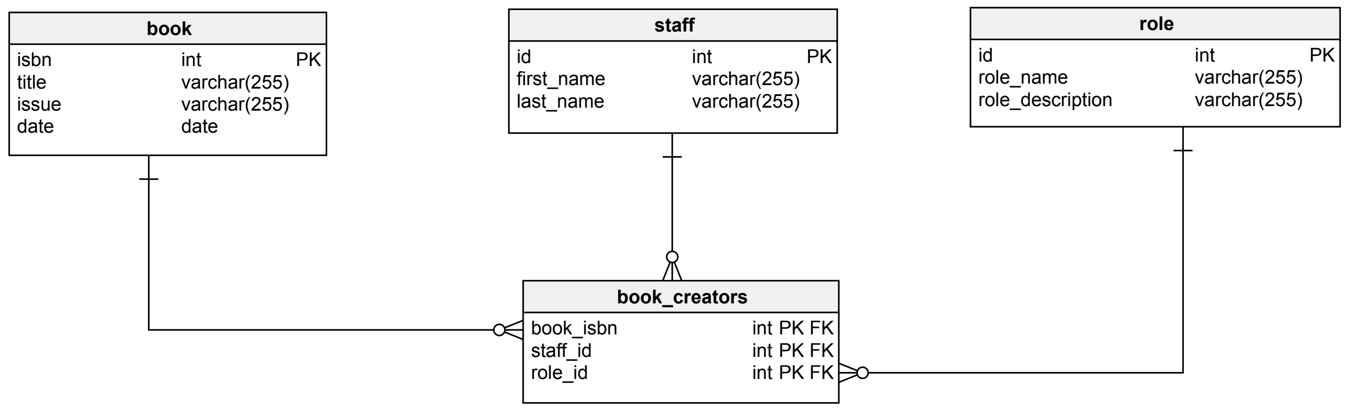
Vertabelo

On to the third entity, `role`! It consists of:

- `id`: The ID of the role; a primary identifier (PI).
- `role_name`: The name of the role.
- `role_description`: A description of that role.

Again, there is a many-to-many relationship between the entities `staff` and `role`. Logic says that one staff member can fill one or many roles when working on a book and that one role can be performed by one or many staff members. When I say role, I mean something like author, co-author, editor, proofreader, translator, illustrator, etc. For instance, the author of one book can also be an illustrator on another book, translator on a third, and proofreader on a fourth.

This example seems even more complicated than the first two. Until now, there were only two tables in the logical model. How will the physical model look in this case? Like this:



EDIT MODEL IN YOUR BROWSER

Does that look scary? Well, maybe a little bit. What happened here is that you were introduced to ternary relationships without knowing it. A ternary relationship is when

The table `book` is important here, it's the starting point from which the creators of the book have to be determined. When I say creators, I mean the staff members who participated in that book as well as their roles in that particular book.

Next, let's analyze the junction table `book_creators`. It has three attributes:

- `book_isbn`: The ISBN of the book; references the `book` table.
- `staff_id`: The ID of the staff member; references the `staff` table.
- `role_id`: The ID of the role; references the `role` table.

The primary key of the table is the unique combination of the attributes `book_isbn`, `staff_id`, and `role_id`.

By creating this junction table, you've solved the many-to-many relationship, which is enough to declare this task finished!

What Are Your Thoughts on Many-to-Many Relationships?

In these three examples, I've tried to show you what many-to-many relationships are in a logical database model. We've also discussed how to approach your physical data model. You've learned that many-to-many relationships in relational databases can be solved by implementing a junction table. And finally, we also demonstrated that you don't have to draw logical and physical models separately. The **VERTABELO DATABASE MODELER** can help you generate a physical model from the logical one, sometimes without any additional tweaks.

Tags: data modeling database design many-to-many relationship

Vertabelo

notified about the latest posts.

Email address

SUBSCRIBE

You may also like

TECHNICAL

Common ER Diagram Mistakes

EXAMPLE ER DIAGRAMS

READ MORE

EXAMPLE ER DIAGRAMS

Tap and Park: A Parking App Data Model

READ MORE

Quick links

Vertabelo Blog

Features

Documentation

Pricing

FAQ

Academy

Vertabelo

Support

If you have any questions or you need our help, you can contact us through our

SUPPORT SITE

Follow Us

Terms of Service Privacy Policy Imprint Investor relations