**Relationships in RDBMS**

Table of Contents

[Introduction 2](#_Toc141309861)

[Types of relations 2](#_Toc141309862)

[One-to-many relation 2](#_Toc141309863)

[Many-to-many relation 3](#_Toc141309864)

[One-to-one relation 5](#_Toc141309865)

# Introduction

One of the most important things in databases is to understand the types of relations in the databases. That stands for both – a process of designing a database model as well as when you’re analyzing your data. Understanding these relations is somehow natural and not so complex but is still essential in the database theory (and practice).

## Types of relations

There are 3 different types of relations in the database:

1. one-to-many
2. many-to-many, and
3. one-to-one

### One-to-many relation

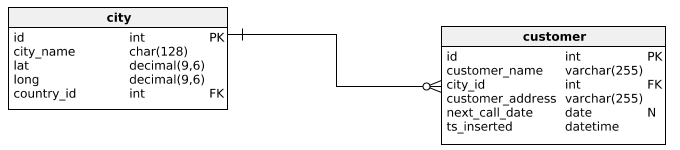
The first of our 3 types of relations, we’ll start with is one-to-many. The reason for that is that it’s the most commonly used and the remaining two are “subtypes” of this one. Let’s start with a real-life problem.

**Example**

Imagine that we want to store a list of all our **customers** in the database.

For each **customer**, we also want to store the **city** where this customer is located, and we know that the **customer will be in exactly one city**.

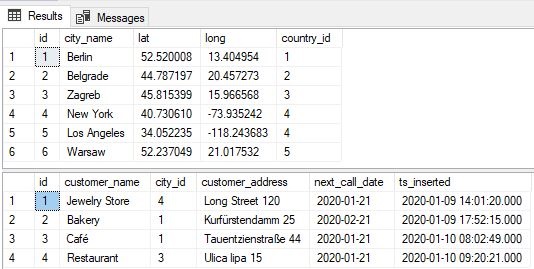
This typical example of one-to-many relation and this is how we solved it in our model:



We simply established a relation from the **city.id** to **customer.city\_id**. And this works, because the customer can be only in one city and the city could have many different customers located in it.

When you want to determine the nature of the relation you need to establish between two tables just do this. In our example – For **one** city, we could have **many** different customers located in it. And the other way around – For **one** customer, we can have only **one** city it’s located in.

So, how to choose between these 3 different types of relations? If you said the word “many” only once, then this is one-to-many relation. If you would use the word “many” two times, the relation would be many-to-many. And if you wouldn’t use it at all, then it would be one-to-one.



We can easily notice few things:

* Not all cites were used (only these with ids 1, 3 and 4 were)
* Each customer had exactly one city it belongs to (customer.city\_id)

### Many-to-many relation

The second out of three types of relations is a many-to-many type. This type is used when both tables could have multiple rows on the other side. Let’s see an example.

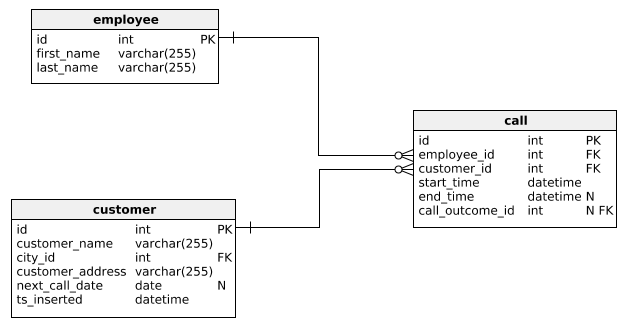
Example

We need to store calls between employees and customers.

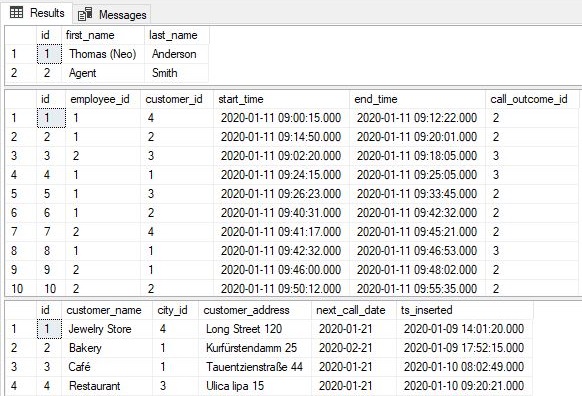
**One** employee, during the time, could call **many** customers. Also, **one** customer, during the time, could receive calls from **many** employees.

Notice that we’ve mentioned the word “many” two times. This is the signal we need to resolve this using many-to-many relation (out of 3 types of relations we have on disposal). To solve it we’ll:

* Add a table between tables **employee** and **customer**
* Add foreign keys (**employee\_id** & **customer\_id**) to that new table (**call**)



Now, when we look from the employee perspective, **one** employee could make **many** (multiple) calls. On the other hand, **one** customer could be related to **many** (multiple) calls. Therefore, many-to-many relation is implemented with adding a new table and one-to-many relations from both sides.



### One-to-one relation

Compared to previously mentioned types of relations, this one is really rarely used. Let’s go with an example.

**Example**

In the database, we want to store employees, but also their valid identity cards. We’re not interested in storing any other types of documents or identity cards that were previously valid, so we need exactly 1 (or none) identity card for 1 employee.

Let’s check this truly is a one-to-one relation. We’ve been given these rules: **One** employee could have only **one** valid identity card in our system. **One** identity card could belong to only **one** employee. We haven’t used the word “many”, so this can’t be any type of relation including the word “many”.

We could do two things here:

* Store identity card details in the **employee** table. This is how it’s usually done and the reason for doing it differently (as mentioned below) is some kind of exception
* Store identity card details in a separate table and relate these two tables with a foreign key. But that foreign key (**identity\_card.employee\_id**), referencing **employee.id**, should, at the same time, be the primary key of the **identity\_card** table. This way we could have only 1 record per employee

We could decide to go with the second option if we want:

* To keep identity card data separately because we want to keep the model clear and follow the same logic in the whole model (each entity from the real-world has its’ own table in the data model)
* Maybe not all employees will have identity cards, so we’ll spare some storage space this way

Please notice that one-to-one was also implemented in the same manner as one-to-many (1 relation) but with the additional condition (the foreign key is also the primary key).