ERD Example: Complexity

by Sophia

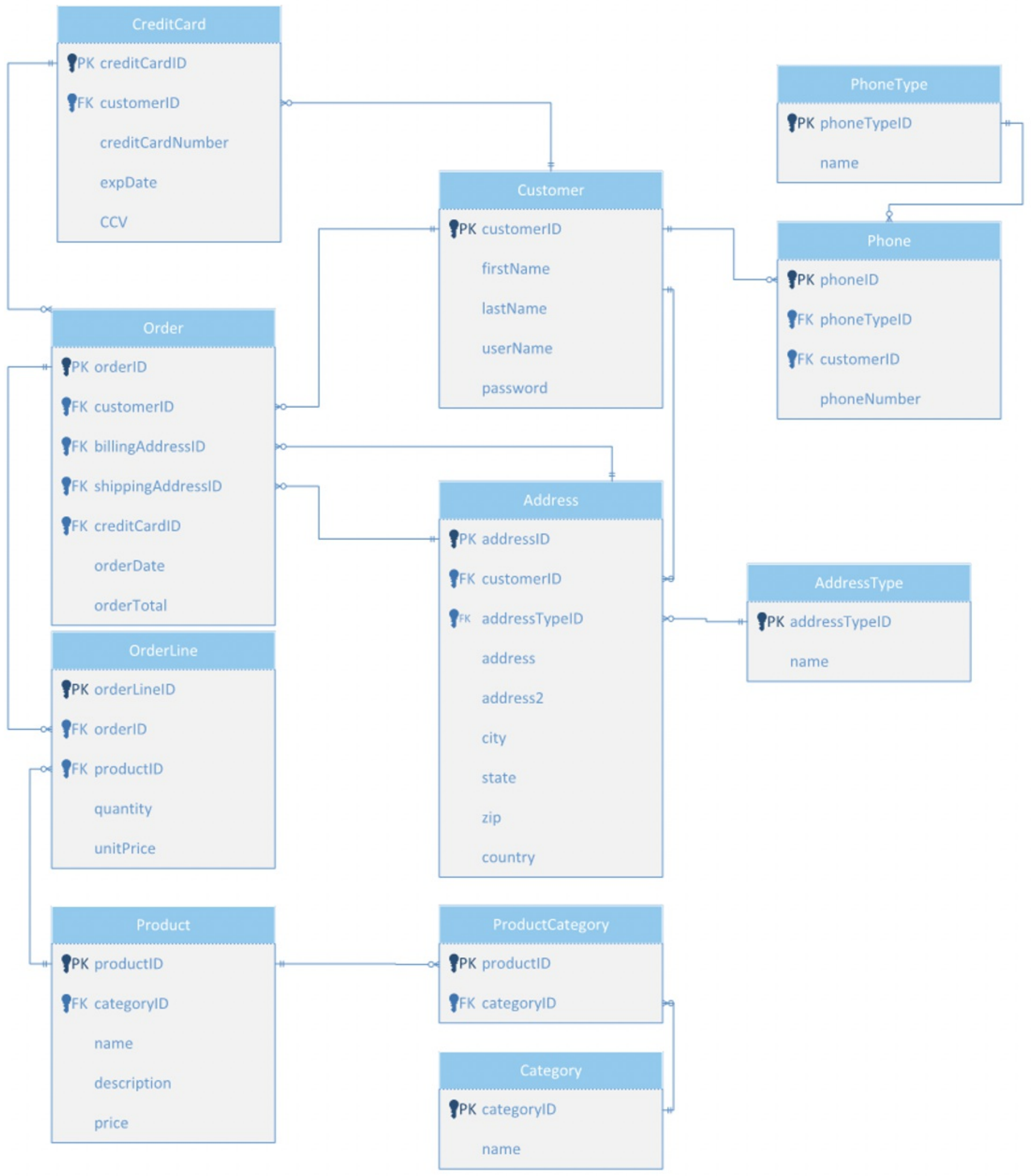
WHAT'S COVERED

This tutorial explores the issues with an overly-complex ERD in two parts:

1. Flexibility Makes Complexity
2. When To Keep It Simple

1. Flexibility Makes Complexity

Look at the database design from the prior tutorial to help us review the issues with an overly-complex ERD:

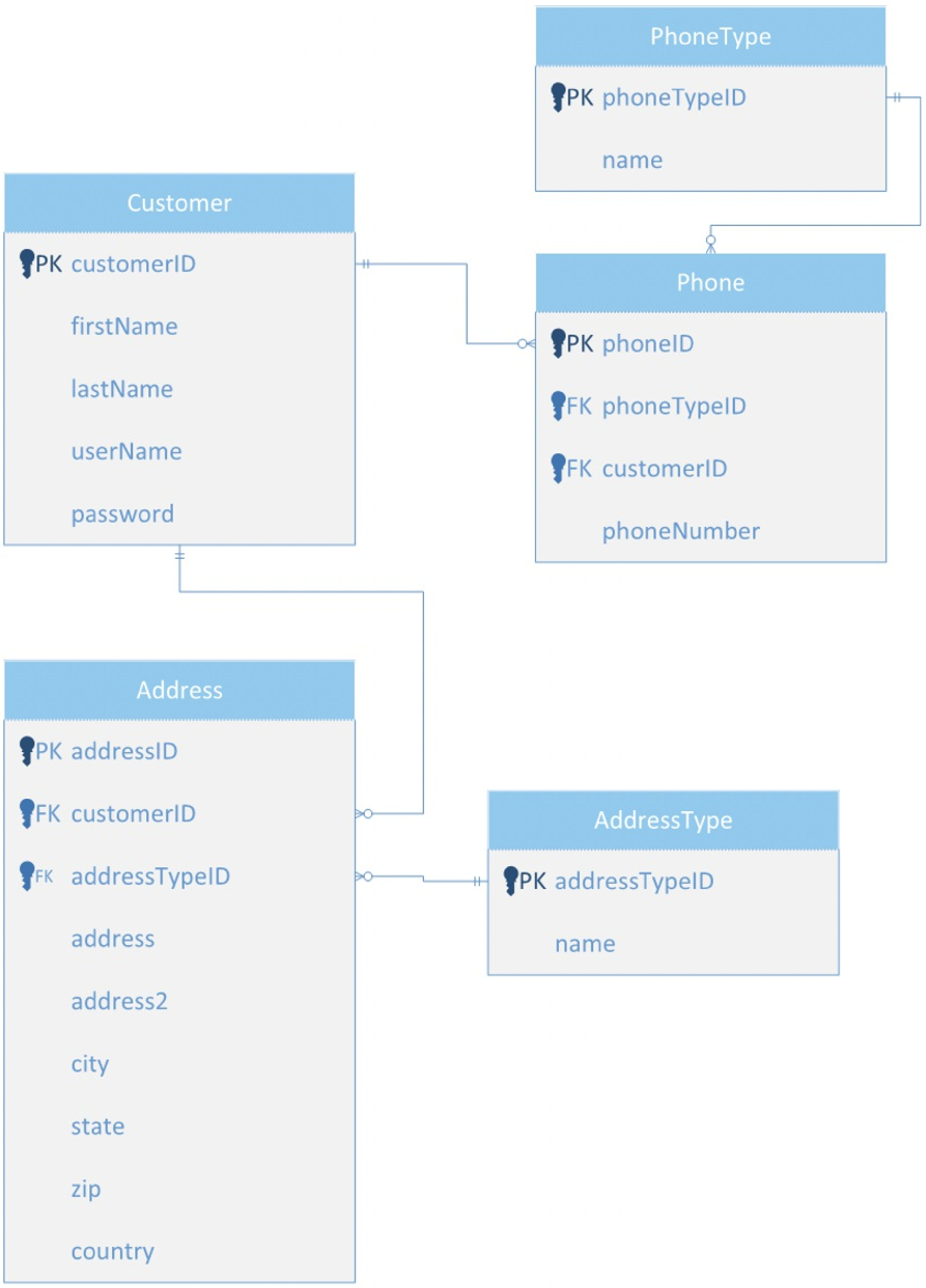


THINK ABOUT IT

If you have an organization that only has a billing and shipping address and just allows a single phone number, does it make sense for the organization to have this level of complex database design?

You have to determine this early on, as the more complex a database design is, the more work it is not only to join the data, but to create the underlying application code to make use of these tables. More validation and checks have to happen to manage and provide information on the data to be more consistent. Let us look just at the customer table to compare a complex and simple data model, given the business rules you have defined.

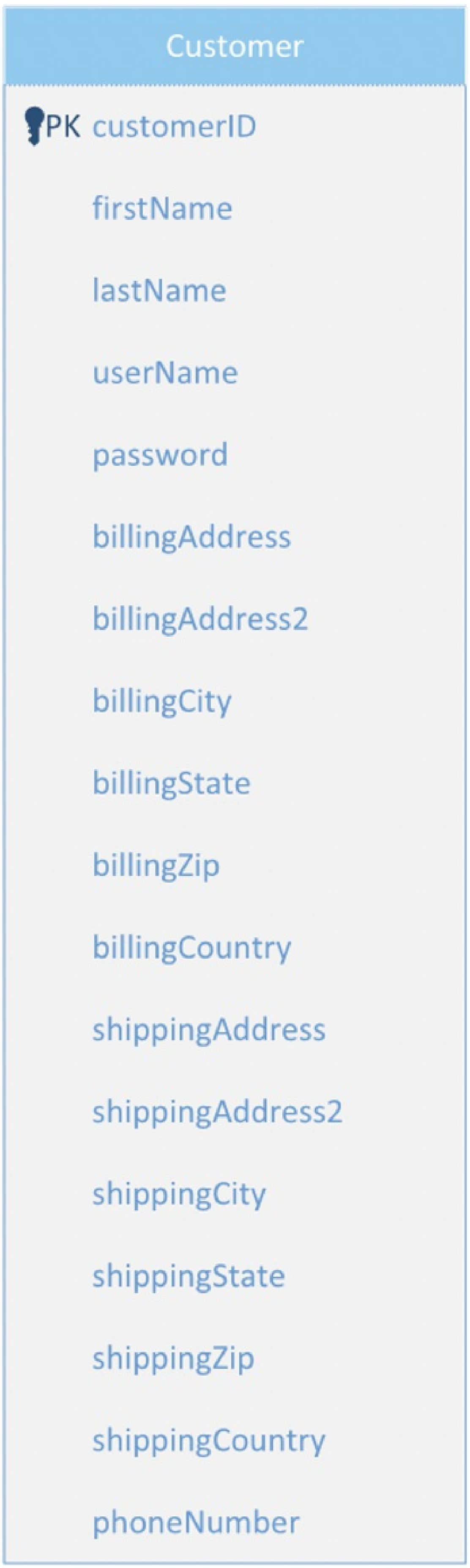
You will see that in the complex design, the customer table is simpler, with separate tables to list phone numbers and addresses allowing as many values as possible. In reality, most customers only store a single shipping address, billing address, and phone number for a vendor.



There can certainly be exceptions, and this type of format will account for that.

2. When To Keep It Simple

However, for a simpler scenario where there is not as much repeat business, a single table could suffice:



Whether to use a simple or complex design will be dependent on the business rules within the organization. If you focus on having full flexibility, you will introduce a lot more complexity not only in the database design but for all other components that link to the database. However, the data model may adapt more easily for scalability than a simpler model would, especially if business rules change, and it will reduce redundancy.

REFLECT

A simpler data model that reduces complexity will make queries, reports, and application code all simpler to create. It allows for faster application development, simpler application code to maintain, and simpler SQL queries. You will see this type of structure to simplify the data model. For example, although in 3NF you would typically split off the zip code, city, and state in its own table, for simplicity with reporting and functionality, it’s commonly just included in the same table as the rest of the address.

SUMMARY

In this tutorial, you learned that **flexibility makes complexity** since more complex designs add more work not only to join the data but to create the underlying application code to make use of all tables included. You also learned that in some cases it makes sense to **keep it simple**. Based on the business need, for example, a simple single table would suffice if there were not much-expected repeat business. Having overly-complex database designs can create a lot more work so it is best to design to the business need.

Source: Authored by Vincent Tran