Logical Design

by Sophia

WHAT'S COVERED

This tutorial explores the next step in designing a new database. We will consider how to extend the conceptual model into a logical model design in two parts:

1. Moving From Concept to a Logical Model
2. Steps To Create A Logical Model

1. Moving From Concept to a Logical Model

The **logical model** extends the conceptual model by defining how the database should be implemented at a high level. The logical model is database agnostic, meaning it is generalized and could be implemented in any database. This model is generally created by business analysts and data architects. The purpose of the logical model is to create a technical map of the rules of the database, as well as to create the structure for the database.

By the end of the logical model design process, all of the entities, attributes, and relationships are defined. The primary key for each entity is specified, as well as the foreign key that links the tables together. Normalization - which you will learn more about in a future tutorial - is also performed at this level. Finally, the characteristics such as location, path, and format should all be included.

TERM TO KNOW

Logical Model
A high-level structure and technical map for a database that specifies primary and foreign keys and has been normalized.

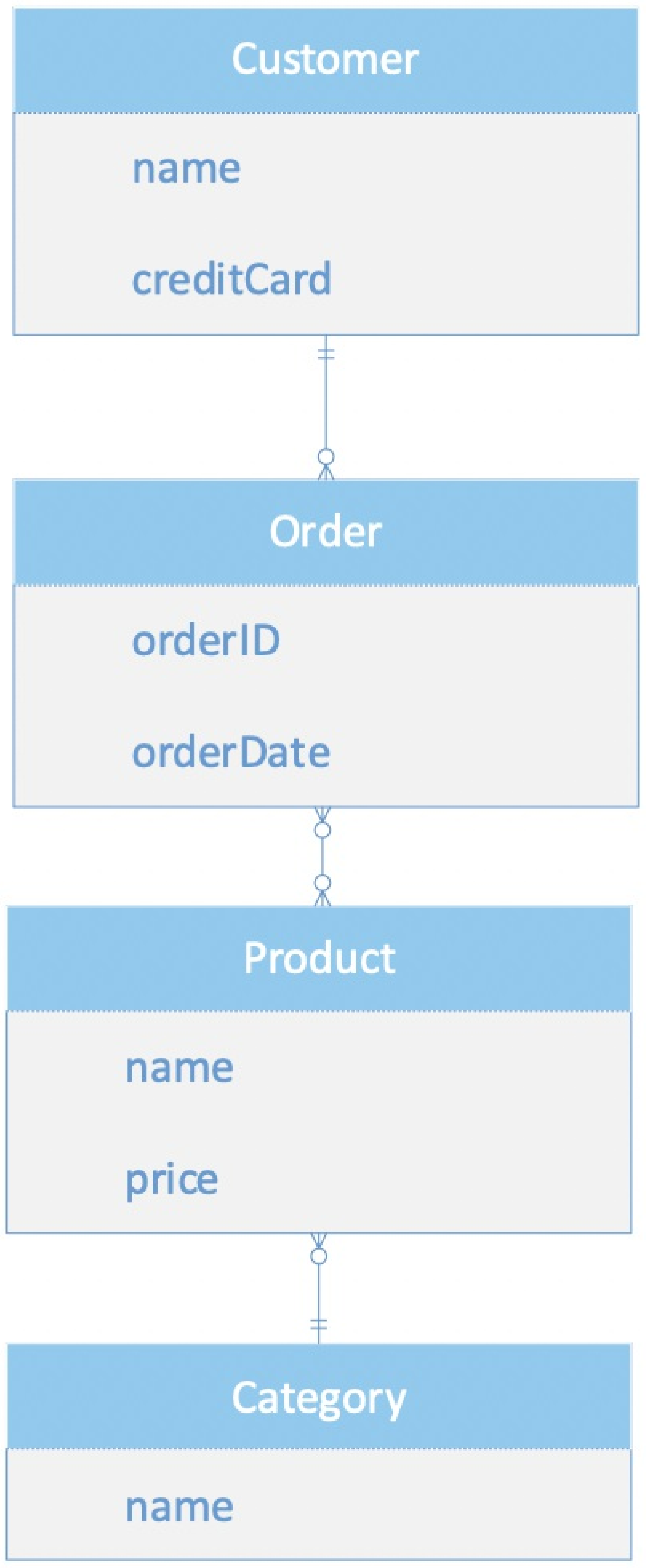
2. Steps To Create A Logical Model

The first step in this process is to map the conceptual model to the logical model. In this process, we will map out the strong entities, supertype/subtypes, weak entities, binary relationships, and higher-degree relationships.

Logical Model Elements	
Strong Entities	An entity that is not dependent on any other entity in the schema. A strong entity will always have a primary key.

Weak Entities	An entity that doesn’t have sufficient attributes to require its own primary key. It depends on a strong entity.
Supertypes/Subtypes	A supertype entity has smaller groups (subtypes) that connect to it but add specialized attributes common to the specific subtype, while the supertype has attributes that are common to all of the subtypes. For example, a supertype might be 'People' with a subtype for 'Employees', 'Vendors', and 'Customers'.
Binary Relationships	The type of relationship between two separate entities. When a binary relationship exists, it can be one-to-one, one-to-many, many-to-one, or many-to-many.
Higher-degree Relationships	A ternary relationship can occur between three entities. Larger numbers of relationships are possible but not recommended.

The strong entities are ones that are on the “one” side of a binary relationship within the data model. Look at the conceptual model from the prior tutorial and see if you can determine which ones are the strong entities:



In this example, the strong entities are the customer and category entities. These are indicated by the use of the two perpendicular crossed lines near the entity which indicate a 'one', as opposed to the three prongs that indicate a 'many' relationship.

Once you have the strong entities mapped out, the second step is to move onto the supertype/subtype relationships and the weak entities. In our example, we don't have any supertype/subtype relationships.

> ↱ EXAMPLE Imagine you have a Pet supertype entity. A subtype could be a Dog or a Cat entity with specific attributes associated with them.

A weak entity is one whose existence is dependent on the strong entity. In the same example of a Dog or a

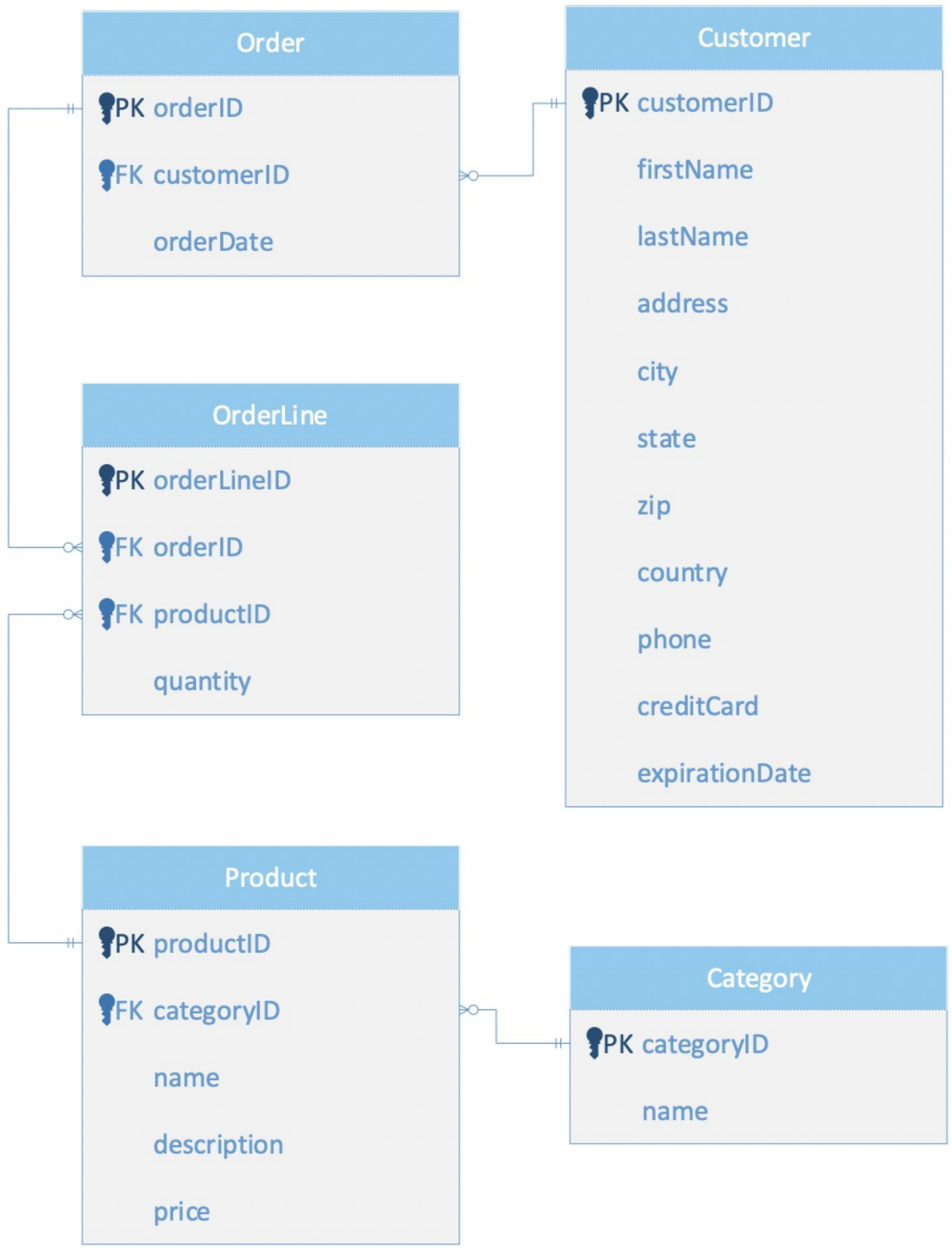
Cat entity, both of those would be weak entities, as they would inherit the primary key from the Pet entity to use as a foreign key. They would also be dependent on the Pet entity to exist; that is, a record in the Dog table could not exist without a related record in the Pet table.

Next, you would work through all of the binary relationships between two entities. For example, you would map out the one-to-many relationships between the product and category tables, as well as the customer and order tables. Once you have those relationships mapped, you would move on to define the relationships between three or more entities, until all of the relationships are defined. In our example, we had a many-to-many relationship between the order and product tables that needed to be resolved. We will get into that level of detail in a future tutorial. For now, make note that for any many-to-many relationship in a conceptual model, it must be resolved by having a bridge in the form of an intermediary table that creates two one-to-many relationships.

At the logical model level, we also define the primary keys and foreign keys as part of the relationships between entities, and validate the model through normalization. We typically also define the integrity constraints of the data, if needed. For example, we may define that the quantity of a numeric value must be an integer and must be greater than zero.

Typically, we won’t define the data types and sizes in the logical model, as this stage of database design is not meant to be database-system-specific. However, it is acceptable to generalize the data types in the logical model. For example, identifying which attributes are a 'number' data type versus a 'text' data type.

Here is an example of the logical data model for the eCommerce company that we have been using:



SUMMARY

The **logical model expands on the conceptual model**to include all entities and relationships between entities. This model includes all attributes for each of the entities, defines the primary key of each entity, and adds the foreign keys to identify the relationship between the entities. Normalization is also performed on the data model during the logical design stage, and any many-to-many binary relationships are handled by splitting them into new one-to-many binary relationships.

Source: Authored by Vincent Tran