

Database Normalization

Copyright 2025 Jordan Miller, adapted from:

[Beginning Database Design: From Novice to Professional, Second Edition](#)

by Clare Churcher Published by [Apress](#), 2012

[Relational Database Design and Implementation, 4th Edition](#)

by Jan L. Harrington Published by [Morgan Kaufmann](#), 2016

Normalization

When we transform our data into relations, this is called **normalization**. There are two ways to do this:

- 1) Create an ER diagram and use simple rules to transform it into relations; then check the relations against theoretical concepts of good database design.
- 2) Use normalization theory to create your relations. (This sometimes results in a better design.)

ER Diagram => Relations

- Create one table for each entity.
- For each entity with **1** or **0...1** multiplicity, choose a **primary key** column; if no column is suitable, use a combination of columns with arbitrary unique identifiers.
- For each entity with multiplicity of **0...*** or **1...***, create **foreign key** columns that contain the **primary keys of the related entities** with multiplicity of 1 or 0...1. (we call these the **parent entities**.)

Also, if this entity has an attribute that is a natural choice for primary key, use that. If not, create one by concatenating the primary key of the parent with some other column to create a unique value.

Normal Forms

- Normal forms are **sets of rules** that a schema must meet in order to be well-designed.

There are six normal forms, which are essentially six levels of refinement that we can bring our schema through, with each increasing level improving the design of the database. In this class, we'll focus on the **first three** normal forms.

First Normal Form (1NF)

- Data is in a two-dimensional table
- There are no multi-valued attributes. (sometimes called "repeating groups")

How could one fix the problem of having multi-valued attributes?

First Normal Form (1NF)

- Data is in a two-dimensional table
- There are no multi-valued attributes. (sometimes called "repeating groups")

"remove the multivalued information from the table. Create a new table with that information and the primary key of the original table."
Churher

First Normal Form (1NF)

Potential problems with data in 1NF:

- Even if we make sure that our tables don't have multi-valued attributes, we may still find that our tables have attributes that aren't strictly related to the entity or make more sense in another entity.

For example, suppose a student table contains an attribute for college phone number. You can imagine how it might cause problems if we were to try to change the phone number of the college. We'd have to update the data for every student!

Second Normal Form (2NF)

Relations are in second normal form if:

- They are in first normal form;
- All non-key attributes are ***functionally dependent*** on ***all*** primary key fields.

Functionally dependent attributes are things you would expect to find when searching for whatever entity is represented by the primary key. They must also be ***uniquely*** determined by the primary key. (meaning the primary key points to only one unique value of that attribute.)

- [Relational Database Design and Implementation, 4th Edition](#) by Jan L. Harrington Published by [Morgan Kaufmann](#), 2016
- [Beginning Database Design: From Novice to Professional, Second Edition](#) by Clare Churcher

Second Normal Form (2NF)

Functional dependency is a one-way relationship. For example, customer first name, last name, email, etc. are all dependent on the customer ID. But the opposite is not necessarily true since more than once customer can have the same last name, etc.

customer_number ->> first_name, last_name, phone

In this example, we say that customer_number is the **determinant**.

Second Normal Form (2NF)

For a schema to be in 2NF, it is not enough for an attribute to be functionally-dependent on only one of the PK columns. It must, in fact, be functionally-dependent on the entire PK, including all PK columns.

Example:

Movie(movie_id, actor_id, actor_name, movie_name)

Both the **movie_id** and **actor_id** are needed to form the PK, because one movie can have many actors, and one actor can be in many movies. (So neither column on its own will have unique values in every row.)

Second Normal Form (2NF)

For a schema to be in 2NF, it is not enough for an attribute to be functionally-dependent on only one of the PK columns. It must, in fact, be functionally-dependent on the entire PK, including all PK columns.

Example:

Movie(movie_id, actor_id, actor_name, movie_name)

Both the **movie_id** and **actor_id** are needed to form the PK, because one movie can have many actors, and one actor can be in many movies. (So neither column on its own will have unique values in every row.) However, the **actor name** is ONLY functionally dependent on the **actor ID**, and the **movie name** is only dependent on the **movie ID**! *Therefore, this relation is NOT in 2NF.*

Second Normal Form (2NF)

To get your relations into 2NF, start by identifying the **determinants** and their **functional dependencies**.

Each determinant will become the primary key of a relation.

Second Normal Form (2NF)

" If a table is not in second normal form, remove those non–key fields that are not dependent on the whole of the primary key. Create another table with these fields and the part of the primary key on which they do depend." Churcher 2012

Remember that there is an art to database design, and if you feel a little lost at this point in your design process, you can draw an ER diagram to see how everything is looking!

Second Normal Form (2NF)

" If a table is not in second normal form, remove those non–key fields that are not dependent on the whole of the primary key. Create another table with these fields and the part of the primary key on which they do depend." Churcher 2012

Movie(movie_id, actor_id, actor_name, movie_name)

becomes...

Actor_instance(movie_id, actor_id)

Actor(actor_id, actor_name)

Movie(movie_id, movie_name)

Second Normal Form (2NF)

" If a table is not in second normal form, remove those non–key fields that are not dependent on the whole of the primary key. Create another table with these fields and the part of the primary key on which they do depend." Churcher 2012


Movie(movie_id, actor_id, actor_name, movie_name)

becomes...

Actor_instance(movie_id, actor_id)

Actor(actor_id, actor_name)

Movie(movie_id, movie_name)



This ought to remind you of the **composite entities** we used to break M:M relationships in ERDs!

Third Normal Form (3NF)

3NF is meant to solve problems that can arise when non-key attributes depend on one another. This is an indication that there is another entity encoded in the relation. Take the following example:

Order (Order_ID, Order_date, Customer_ID, Customer_Name, Customer_Address)

As you can see, this is in 2NF because all these attributes are functionally dependent on the order ID. However, you'll notice that Customer_Name and Customer_Address are also dependent on Customer_ID.

Third Normal Form (3NF)

example:

Order (Order_ID, Order_date, Customer_ID, Customer_Name, Customer_Address)

Why is it a problem that these customer attributes depend on one another? Suppose the **Order table** contains several orders from the same customer. If this customer were to change addresses, we would have to update all these orders. **Customer** should probably be its own entity, separate from **Order**.

Third Normal Form (3NF)

example:

Order (Order_ID, Order_date, **Customer_ID**)

Customer (Customer_ID, Customer_Name, Customer_Address)

We've pulled **Customer** into a separate relation, and referred to it in **Order** using a foreign key.

Third Normal Form (3NF)

example:

Order (Order_ID, Order_date, **Customer_ID**)

Customer (Customer_ID, Customer_Name, Customer_Address)

We've pulled **Customer** into a separate relation, and referred to it in **Order** using a foreign key.

*These relationships between non-key attributes are sometimes called **transitive dependencies**.*

Normalization Summary

- 1NF: If an attribute with multiple values is present, create a new table with attribute and the primary key of the original
- 2NF: If the table has an attribute that is not functionally dependent on all primary key columns, move it to a new table. Its determinant will form the primary key for the new table.
- 3NF: If the table has an attribute that is functionally dependent on some attribute that is not the primary key, move it and its determinant to a new table; the determinant will form the new primary key of that table.