Normalización

La normalización es un proceso paso a paso con el que podemos asegurarnos de que nuestros esquemas cumplan con esas reglas, y así reducir los posibles problemas a futuro.

* Su objetivo principal es reducir la redundancia en los datos.

Es decir, reducir la cantidad de datos repetidos entre registros.

* Se divide en formas normales ( o FNs), que son básicamente los pasos de revisión y mejora que debemos de seguir al diseñar nuestros esquemas.

En total existen 5 formas normales, pero en esta lección nos enfocaremos en las 3 primeras y más importantes. La 4FN y 5FN solo aplican para casos muy específicos.

* Algo muy curioso es que la mayoría de desarrolladores experimentados ya no recuerdan cómo diferenciar estas formas normales.

Solo saben cómo se aplican sus reglas en conjunto “*a puro ojo*”

**Primer Forma Normal (1FN)**

La 1FN se basa en **2 reglas** principales:

1. Todos los registros deben de tener una llave primaria **única**.

| **ID** | **Name** | **Email** |
| --- | --- | --- |
| 1 | Alice | [alice@email.com](mailto:alice@email.com) |
| 1 | Alice | [alicebackup@email.com](mailto:alicebackup@email.com) |

1. Las columnas deben contener datos **atómicos**, **indivisibles**.
   * No debemos de tener columnas que tengan **multiples datos**.

| **Phone Numbers** |
| --- |
| 83645746, 60324573, 70985647 |
| 60784354, 60312343, 7045342 |

Ni **multiples columnas** para una misma categoría de dato.

| **Phone Number 1** | **Phone Number 2** | **Phone Number 3** |
| --- | --- | --- |
| 83645746 | 60324573 | 70985647 |
| 60784354 | 60312343 | 7045342 |

En caso de no cumplir con estas reglas, esos datos deben de moverse a una **nueva tabla** que se relacione con la original.

Veamos este ejemplo:

* Tenemos una tabla con información de automóviles.
  + Y los automóviles pueden tener varios dueños.

**Cars**

| **Id** | **Plate Num.** | **Make** | **Model** | **Type** | **Owners** |
| --- | --- | --- | --- | --- | --- |
| 1 | BCA895 | Honda | Civic | Coupe | John Doe, Luke Skywalker |
| 2 | KND375 | Ford | Mustang | Coupe | Jane Doe |
| 3 | KJM876 | Honda | Civic | Coupe | John Doe, Luke Skywalker, Jane Doe |
| 4 | HJD535 | Ford | Raptor | Truck | Luke Skywalker |

El problema es que en la tabla estamos guardando los dueños en una columna divididos por coma.

* + Los datos guardados de esta forma llevan a **muchos problemas**, y requieren que se dividan manualmente en el código.
  + Además, **imagínate** que se tratase de notas o descripciones. *Dividir con comas sería imposible.*

Una solución (*hipotética*) sería tener distintos registros para cada dueño:

Cars (También Incorrecta)

| **Id** | **Plate Num.** | **Make** | **Model** | **Type** | **Owner** |
| --- | --- | --- | --- | --- | --- |
| 1 | BCA895 | Honda | Civic | Coupe | John Doe |
| 1 | BCA895 | Honda | Civic | Coupe | Luke Skywalker |
| 2 | KND375 | Ford | Mustang | Coupe | Jane Doe |
| 3 | KJM876 | Honda | Civic | Coupe | John Doe |
| 3 | KJM876 | Honda | Civic | Coupe | Luke Skywalker |
| 3 | KJM876 | Honda | Civic | Coupe | Jane Doe |
| 4 | HJD535 | Ford | Raptor | Truck | Luke Skywalker |

Sin embargo, aquí rompemos la regla de que las llaves primarias deben ser **siempre** únicas.

* + Sería incluso imposible de lograr en un escenario real. El **motor** no nos dejaría.

Otra solución (*la más común, pero también incorrecta*) seria tener multiples columnas para cada dueño:

Cars (También Incorrecta)

| **Id** | **Plate Num.** | **Make** | **Model** | **Type** | **Owner 1** | **Owner 2** | **Owner 3** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | BCA895 | Honda | Civic | Coupe | John Doe | Luke Skywalker |  |
| 2 | KND375 | Ford | Mustang | Coupe | Jane Doe |  |  |
| 3 | KJM876 | Honda | Civic | Coupe | John Doe | Luke Skywalker | Jane Doe |
| 4 | HJD535 | Ford | Raptor | Truck | Luke Skywalker |  |  |

Acá tenemos varios registros con varios valores nulos, lo cual no es ideal.

Además **no es escalable**. ¿Qué pasa si hay un automóvil con 4 dueños?

La solución es simple. Debemos de separar esos **potenciales valores multiples** en otra tabla relacionada a la principal:

**Cars (Correcta)**

| **Id** | **Plate Num.** | **Make** | **Model** | **Type** |
| --- | --- | --- | --- | --- |
| 1 | BCA895 | Honda | Civic | Coupe |
| 2 | KND375 | Ford | Mustang | Coupe |
| 3 | KJM876 | Honda | Civic | Coupe |
| 4 | HJD535 | Ford | Raptor | Truck |

**Owners**

| **Id** | **Name** |
| --- | --- |
| 1 | John Doe |
| 2 | Luke Skywalker |
| 3 | Jane Doe |

¿Ahora, cómo las relacionamos?

* En este caso, sabemos que un carro puede tener varios dueños,
  + Y cada dueño puede tener a la vez varios carros.
* Entonces, como vimos en [Many to Many (N:N)](https://www.notion.so/Many-to-Many-N-N-6715f51ca09f47408a51b1577c02fada?pvs=21), necesitamos también una tabla cruz para unir nuestras 2 tablas.

**CarsOwners**

| **Id** | **Car Id** | **Owner Id** |
| --- | --- | --- |
| 1 | 1 | 1 |
| 2 | 1 | 2 |
| 3 | 2 | 3 |
| 4 | 3 | 1 |
| 5 | 3 | 2 |
| 6 | 3 | 3 |
| 7 | 4 | 2 |

**Segunda Forma Normal (2FN)**

La 2FN busca eliminar las dependencias parciales.

* Una dependencia parcial solo puede ocurrir en tablas donde la llave primaria está conformada por más de una columna (*llaves compuestas*).
  + Hasta ahora hemos recomendado siempre usar un id numérico como llave primaria.
  + Sin embargo, hay algunos casos donde uno puede usar dos columnas en conjunto como la llave primaria. Por ejemplo, en las tablas cruz.
* Si es una tabla que usa un id numérico como llave primaria (como lo hemos recomendado), y ya cumple con 1FN, **automáticamente** cumple con 2FN.
  + Es decir que la gran mayoría de tablas que creemos van a cumplir con 2FN una vez cumplan con 1FN.

Teniendo esto claro, veamos un ejemplo con una tabla de llave primaria compuesta:

Todo dato que no dependa de la llave primaria **completa** para ser identificado es considerado una dependencia parcial. Si alguno de estos datos puede ser identificado gracias a otro dato **además** de la llave primaria, se mueve a otra tabla aparte.

**Pasos para aplicar 2FN correctamente:**

1. Identificar si la tabla tiene una llave primaria compuesta.
2. Eliminar los campos que no dependen de la clave primaria completa.
3. Mover todos los datos repetidos entre varios registros a sus propias tablas.

**Tabla original con dependencia parcial**

Antes de aplicar la 2FN, observemos la estructura de la tabla original:

| **Student ID** | **Subject** | **Student Name** | **Phone Number** |
| --- | --- | --- | --- |
| 1 | Math | Alice | 12345 |
| 1 | Science | Alice | 12345 |
| 1 | Math | Alice | 67890 |
| 1 | Science | Alice | 67890 |
| 2 | English | Bob | 54321 |
| 2 | History | Bob | 54321 |
| 3 | Science | Charlie | 98765 |
| 3 | Science | Charlie | 43210 |
| 3 | Science | Charlie | 11223 |

En esta tabla, la llave primaria está compuesta por Student ID y Subject, ya que cada estudiante puede estar inscrito en múltiples materias, y una misma materia puede ser tomada por varios estudiantes. Esta combinación garantiza que cada fila en la tabla sea única.

* Sin embargo, podemos notar que la columna Phone Number no depende de esta llave compuesta, sino únicamente de Student ID.
* Esto significa que Phone Number tiene una dependencia parcial, ya que su valor no requiere de Subject para ser determinado.
* Esta dependencia parcial viola la Segunda Forma Normal (2FN), lo que indica que debemos realizar modificaciones en la estructura de la tabla.

**Aplicación de la Segunda Forma Normal (2FN) con un ejemplo:**

Para normalizar esta tabla y aplicar la 2FN, realizamos los siguientes pasos:

1. **Identificamos la dependencia parcial**: El Phone Number no depende de la combinación Student ID y Subject, sino solo de Student ID.
2. **Eliminamos los campos que no dependen completamente de la clave primaria**: El Phone Number debe moverse a una tabla separada.

**Nueva tabla Students (eliminando la dependencia parcial)**

| **Student ID** | **Student Name** |
| --- | --- |
| 1 | Alice |
| 2 | Bob |
| 3 | Charlie |

**Nueva tabla StudentPhones (separando los teléfonos)**

| **Student ID** | **Phone Number** |
| --- | --- |
| 1 | 12345 |
| 1 | 67890 |
| 2 | 54321 |
| 3 | 98765 |
| 3 | 43210 |
| 3 | 11223 |

**Nueva tabla StudentSubjects (manteniendo la relación entre estudiantes y materias)**

| **Student ID** | **Subject** |
| --- | --- |
| 1 | Math |
| 1 | Science |
| 2 | English |
| 2 | History |
| 3 | Science |

Con esta separación, ahora todas las dependencias están correctamente distribuidas, y la base de datos cumple con la 2FN al eliminar las dependencias parciales.

**Tercer Forma Normal (3FN)**

La 3FN busca eliminar las dependencias transitivas.

Una dependencia transitiva es todo dato que no dependa **directamente** de la llave primaria para ser identificado.

* Es decir, todo dato que pueda ser identificado por otra columna que no sea el primary key.
* *Esto suena muy extraño, pero tiene mucho sentido una vez que lo ponemos a prueba:*

Sigamos utilizando el ejemplo de la tabla **Cars**.

**Cars**

| **Id** | **Plate Num.** | **Make** | **Model** | **Type** |
| --- | --- | --- | --- | --- |
| 1 | BCA895 | Honda | Civic | Coupe |
| 2 | KND375 | Ford | Mustang | Coupe |
| 3 | KJM876 | Honda | Civic | Coupe |
| 4 | HJD535 | Ford | Raptor | Truck |

Si analizamos columna por columna:

* El Plate Num. puede ser identificado de manera única **solamente** por el Id.
  + Pueden haber carros con el mismo Make, Model y/o Type, pero distintas placas (como los registros 1 y 3).
  + La única forma de identificar de manera única cada placa es usando el **Id**.
* Para cumplir con la 3FN, Make, Model, y Type también deberían depender únicamente del Id **para ser identificados.**
* Sin embargo, puedes deducir estas características con el Plate Num..
  + *La placa siempre va a ser para un mismo modelo.*
* Debido a esto, lo mejor sería que estos datos estén en otra tabla por aparte.

**Cars**

| **Id** | **Plate Num.** | **Model Id** |
| --- | --- | --- |
| 1 | BCA895 | 1 |
| 2 | KND375 | 2 |
| 3 | KJM876 | 1 |
| 4 | HJD535 | 3 |

**Models**

| **Id** | **Make** | **Name** | **Type** |
| --- | --- | --- | --- |
| 1 | Honda | Civic | Coupe |
| 2 | Ford | Mustang | Coupe |
| 3 | Honda | Civic | Coupe |
| 4 | Ford | Raptor | Truck |

Y ya está resuelto?

* No realmente… Si analizamos la tabla de Models, veremos que aún no cumple con 3FN.

**Models**

| **Id** | **Make** | **Name** | **Type** |
| --- | --- | --- | --- |
| 1 | Honda | Civic | Coupe |
| 2 | Ford | Mustang | Coupe |
| 3 | Honda | Civic | Coupe |
| 4 | Ford | Raptor | Truck |

* También podemos identificar el Make de un carro gracias al Model.
  + Un Civic **siempre** será Honda, así como un Mustang **siempre** será Ford.

**Models**

| **Id** | **Make Id** | **Name** | **Type** |
| --- | --- | --- | --- |
| 1 | 1 | Civic | Coupe |
| 2 | 2 | Mustang | Coupe |
| 3 | 1 | Civic | Coupe |
| 4 | 2 | Raptor | Truck |

Makes

| **Id** | **Name** |
| --- | --- |
| 1 | Honda |
| 2 | Ford |

* Asimismo, también podemos identificar el Type gracias al Model.
  + Existen excepciones, pero normalmente los modelos tienen los mismos estilos siempre.
  + No existe un Civic Truck.

**Models**

| **Id** | **Make Id** | **Name** | **Type Id** |
| --- | --- | --- | --- |
| 1 | 1 | Civic | 1 |
| 2 | 2 | Mustang | 1 |
| 3 | 1 | Civic | 1 |
| 4 | 2 | Raptor | 2 |

**Makes**

| **Id** | **Name** |
| --- | --- |
| 1 | Honda |
| 2 | Ford |

**Types**

| **Id** | **Name** |
| --- | --- |
| 1 | Coupe |
| 2 | Truck |

**Resultado final**

**Tabla original**

**Cars**

| **Id** | **Plate Num.** | **Make** | **Model** | **Type** | **Owners** |
| --- | --- | --- | --- | --- | --- |
| 1 | BCA895 | Honda | Civic | Coupe | John Doe, Luke Skywalker |
| 2 | KND375 | Ford | Mustang | Coupe | Jane Doe |
| 3 | KJM876 | Honda | Civic | Coupe | John Doe, Luke Skywalker, Jane Doe |
| 4 | HJD535 | Ford | Raptor | Truck | Luke Skywalker |

**Tablas normalizadas**

**Cars**

| **Id** | **Plate Num.** | **Model Id** |
| --- | --- | --- |
| 1 | BCA895 | 1 |
| 2 | KND375 | 2 |
| 3 | KJM876 | 1 |
| 4 | HJD535 | 3 |

**Models**

| **Id** | **Make Id** | **Name** | **Type Id** |
| --- | --- | --- | --- |
| 1 | 1 | Civic | 1 |
| 2 | 2 | Mustang | 1 |
| 3 | 2 | Raptor | 2 |

**Makes**

| **Id** | **Name** |
| --- | --- |
| 1 | Honda |
| 2 | Ford |

**Types**

| **Id** | **Name** |
| --- | --- |
| 1 | Coupe |
| 2 | Truck |

**Owners**

| **Id** | **Name** |
| --- | --- |
| 1 | John Doe |
| 2 | Luke Skywalker |
| 3 | Jane Doe |

**Cars Owners**

| **Id** | **Car Id** | **Owner Id** |
| --- | --- | --- |
| 1 | 1 | 1 |
| 2 | 1 | 2 |
| 3 | 2 | 3 |
| 4 | 3 | 1 |
| 5 | 3 | 2 |
| 6 | 3 | 3 |
| 7 | 4 | 2 |

* **Ya no hay redundancia.** 
  + **No tenemos datos repetidos en ninguna tabla.**

**Segundo Ejemplo 3FN.**

**Explicación del problema**

En la versión **incorrecta**, la tabla **Órdenes** contiene datos sobre productos y sus precios, lo que introduce una **dependencia transitiva**. En este caso:

* Precio depende del Producto.
* Producto depende del Código de Orden.

El problema aquí es que si el precio de un producto cambia, tendríamos que actualizar múltiples filas en la tabla **Órdenes**, lo que genera **redundancia** y posibles **inconsistencias** en los datos.

**Ejemplo de estructura incorrecta**

**Órdenes (Incorrecto)**

| **Id** | **Código de Orden** | **Producto** | **Cantidad** | **Precio** |
| --- | --- | --- | --- | --- |
| 1 | PO-3425 | Bola de baloncesto | 2 | 5.99 |
| 2 | PO-3254 | Desinfectante | 1 | 8.99 |
| 3 | PO-4356 | Bola de baloncesto | 1 | 5.99 |

**Aplicación de la 3FN**

Para eliminar esta dependencia transitiva, la solución correcta es **separar los productos en una tabla independiente**:

1. **Se eliminan la columna Producto y Precio de la tabla Órdenes** porque Producto no es una propiedad de la orden, sino un atributo propio de los productos.
2. **Se crea la tabla Productos**, donde cada producto tiene su propio Id, Nombre y Precio.
3. **Las tablas se relacionan a través del Id del producto**, evitando redundancias y asegurando que cualquier cambio en los datos de un producto solo requiera una única modificación.

**Ejemplo de estructura correcta**

Para reflejar correctamente la relación entre órdenes y productos, agregamos una **tabla intermedia** (Orden\_Productos). Esta tabla es necesaria porque en un sistema real:

* Una **orden** puede contener **varios productos**.
* Un **producto** puede aparecer en **varias órdenes**.

La tabla Orden\_Productos evita redundancias y garantiza una estructura **normalizada y escalable**.

**Órdenes (Correcto)**

Esta tabla solo almacena información general de la orden.

| **Id** | **Código de Orden** | **Cantidad** |
| --- | --- | --- |
| 1 | PO-3425 | 2 |
| 2 | PO-3254 | 1 |
| 3 | PO-4356 | 1 |

**Productos**

Cada producto tiene un identificador único y sus atributos propios como el nombre y el precio.

| **Id** | **Nombre** | **Precio** |
| --- | --- | --- |
| 1 | Bola de baloncesto | 5.99 |
| 2 | Desinfectante | 8.99 |

**Orden\_Productos (Tabla intermedia)**

Esta tabla se encarga de conectar las órdenes con los productos, permitiendo registrar cuántos productos específicos hay en cada orden.

| **Id** | **Código de Orden** | **Producto ID** | **Cantidad** |
| --- | --- | --- | --- |
| 1 | PO-3425 | 1 | 2 |
| 2 | PO-3254 | 2 | 1 |
| 3 | PO-4356 | 1 | 1 |

**Beneficios de aplicar 3FN**

Con esta separación, los datos cumplen con la **3FN** porque:

* Cada atributo no clave en la tabla **Órdenes** depende solo de la clave primaria (Código de Orden).
* La información de los productos ahora reside en una tabla aparte (Productos), sin dependencias transitivas.

Este diseño facilita la **consistencia, integridad y eficiencia** de la base de datos.

**Normalización “alternativa”**

La normalización de bases de datos es fundamental para diseñar estructuras eficientes, pero a menudo puede parecer ambigua o confusa. Muchos desarrolladores experimentados no recuerdan cada forma normal de memoria, ya que con la práctica se desarrolla una intuición que permite identificar estructuras bien diseñadas sin necesidad de revisar reglas formales.

Este enfoque práctico resume la normalización en cuatro pasos clave para ayudar a detectar y corregir problemas en el diseño de una base de datos.

**Pasos clave para una buena normalización**

La normalización de bases de datos puede abordarse de manera práctica siguiendo algunos principios básicos. A continuación, se presentan cuatro pasos esenciales para garantizar un diseño estructurado y eficiente, sin necesidad de recordar todas las formas normales exactas.

**1. Cero redundancia**

* No debe haber datos repetidos innecesariamente en una tabla.
* Si detectas string repetidos en varias filas, es señal de que deben estar en una tabla aparte.
* La solución es separar esos datos en otra tabla relacionada con la original mediante una relación 1:N.

**2. Cada tabla debe tener una clave primaria**

* Toda tabla necesita una identificación única, ya sea una clave primaria natural (como un código o un identificador único) o una clave primaria artificial (como un ID autoincremental).
* Esto garantiza que cada fila sea distinguible de las demás.

**3. Relaciones entre tablas deben ser lógicas**

* Revisa si la estructura de tus tablas refleja correctamente las relaciones del mundo real.
* Si una tabla tiene muchos datos similares en varias filas, puede ser mejor dividirla en una relación 1:N.
* Para relaciones muchos a muchos (N:N), usa una tabla intermedia.

**4. Evita dependencias indirectas o transitivas**

* Si un dato depende de otro campo que no es la clave primaria, revisa si debería estar en otra tabla.
* Un caso común es cuando una tabla almacena datos que dependen de una categoría, en lugar de estar normalizados en una estructura aparte.

**Comparación con la normalización tradicional**

La normalización tradicional sigue formas normales bien definidas (1NF, 2NF, 3NF). Este enfoque práctico es útil cuando quieres aplicar las reglas sin entrar en demasiados tecnicismos. Sin embargo, en bases de datos críticas y de gran escala, seguir las formas normales puede evitar problemas a largo plazo.

**Errores comunes al aplicar esta regla de forma simplificada**

* **Crear demasiadas tablas:** Dividir en exceso puede hacer que el diseño sea difícil de manejar y poco eficiente.
* **No identificar bien las relaciones:** Es importante analizar si una relación es realmente 1:N o si necesita una estructura N:N.
* **Ignorar dependencias ocultas:** Asegúrate de que los datos se almacenen de manera que no dependan de campos que no sean la clave primaria.

Este método te permitirá normalizar bases de datos de manera efectiva sin necesidad de recordar todas las formas normales exactas. Con práctica, desarrollarás el "ojo entrenado" para estructuras bien diseñadas