**¿Qué es normalización?**

La normalización de la base de datos es simplemente una convención para dividir tablas grandes de datos en tablas separadas más pequeñas con el objetivo principal de no repetir datos. ¿Por qué es esto tan importante? Digamos que usa un reloj para poder ver la hora, porque es muy importante que sepa cuál es la hora actual. ¿Usar ocho relojes lo haría más fácil? ¡De ninguna manera! Ahora tenemos ocho cuentas contradictorias de cuál es el momento adecuado. Peor aún, si alguna vez queremos actualizar la hora, tendríamos que hacerlo para cada reloj de forma independiente. ¡Eso no es muy eficiente!

Puedes aplicar un concepto similar al diseño de la base de datos. Si queremos almacenar la dirección de correo electrónico de un usuario, queremos almacenarla en un solo lugar. Entonces, si alguna vez necesitamos referirnos a él nuevamente, simplemente usamos la identificación. La identificación nunca cambiará, por lo que incluso si actualizamos la dirección de correo electrónico del usuario, ninguna de las otras conexiones que definimos en nuestra base de datos se dañará. ¡Ordenado!

A continuación se presentan las tres reglas principales de normalización de la base de datos. Debes usarlos como guía para diseñar tus ERD. Sin embargo, recuerde siempre que son convenciones comunes y no reglas absolutas. Es posible llevar la normalización al extremo. Por ejemplo, un campo de dirección simple. Un estado puede tener muchas ciudades, una ciudad puede tener muchas calles, una calle puede tener muchos edificios, un edificio puede tener muchos apartamentos, un apartamento puede tener muchos residentes ... y así sucesivamente.   ¡Eso puede volverte realmente loco muy rápido! En las siguientes secciones, aprenderás más sobre por qué este tipo de complejidad puede ser ineficiente, especialmente para tareas simples.

***Primera forma***

Cada columna de su tabla solo puede tener 1 valor.

Ex. No debes tener una columna de dirección en tu tabla que enumere la dirección, la ciudad, el estado y el código postal, todos separados por comas.

***Segunda forma***

Cada columna de tu tabla que no sea una clave (primaria o extranjera) debe tener valores únicos.

Ex. Si tienes una tabla de películas con una columna de categorías, no debes repetir una categoría más de una vez.

***Tercera forma***

No puedes tener una columna sin clave que dependa de otra columna sin clave.

Ex. Si tienes una tabla de libros con columnas publisher\_name y publisher\_address, publisher\_address y publisher\_name deben separarse en una tabla separada y vincularse a los libros con una clave foránea. Publisher\_address depende de publisher\_name y ninguna de las columnas es una columna clave.

**Pautas**

En el futuro, puede que te encuentres trabajando con una empresa que ha configurado sus convenciones de bases de datos de manera un poco diferente, pero estas son las pautas que creemos que son las mejores para este curso:

1. **Haz** que el nombre de la tabla sea plural y TODO minúscula: házlo en plural (por ejemplo, usuarios, clientes\_potenciales, sitios, clientes, capítulos, cursos, módulos)
2. **Usa** "id" como la clave principal; asígnale el nombre id (también haga que se incremente automáticamente).
3. **Nombra** claves foráneas con singular\_table\_name\_id al hacer referencia a una clave primaria en otra tabla nombrala de esta manera [nombre singular de la tabla a la que se refiere] \_id (por ejemplo, user\_id, lead\_id, site\_id, client\_id, chapter\_id, course\_id, module\_id).
4. **Usa** created\_at y updated\_at como columnas para la marca de tiempo en CADA tabla que cree.

Cuando hacemos cosas en ORM o en Ruby on Rails, se vuelve extremadamente importante que sigamos estas convenciones de nomenclatura.

**Simple de datos simples:**

**VARCHAR** (número de caracteres)

Se usa para almacenar valores no numéricos que pueden tener hasta 255 caracteres. Se llama VARCHAR porque puede almacenar un número variable de caracteres y solo usará el espacio requerido para cada registro almacenado en la base de datos. VARCHAR debería usarse para valores con diferentes longitudes de caracteres, como un correo electrónico, nombre o apellido.

**CHAR** (número de caracteres)

También se usa para almacenar valores no numéricos, sin embargo, utilizará todo el espacio para el número de caracteres establecido independientemente de qué valor se agregue a el. Por ejemplo, si configuró CHAR (15) e intento almacenar el valor "coding", usará el equivalente a 15 caracteres aunque "coding" tenga solo 6 caracteres. Char es bueno para usar en cosas que siempre tendrán un número determinado de caracteres. Char funcionaría bien para algo como una abreviatura de estado.

**INT**

Se usa para almacenar enteros.

Las columnas que encontrará principalmente utilizando el INT son cosas como un identificador único para cada tabla. La mayoría de las filas en una tabla no excederá de 2,1 mil millones de registros. INT es bueno para usar para la mayoría de los valores numéricos normales, como un número de teléfono o un código postal.

**sin signo**(solo números positivos): puede almacenar valores numéricos desde 0 hasta 4294967295

**con signo** (números positivos y negativos): puede almacenar valores numéricos desde -2147483648 hasta 2147483647

**BIGINT**

BIGINT se usaría para columnas que tendrían que almacenar grandes cantidades. En la mayoría de los casos, no necesitarías BIGINT, pero si quisieras almacenar algo como una identificación de Facebook cuando utilizas la API de Facebook, dado que tienen más de mil millones de usuarios, la identificación deberá ser un tipo de datos de BIGINT.

**sin signo** (solo números positivos nuevamente): puede almacenar valores numéricos desde 0 hasta 18446744073709551615

**con signo** (números positivos y negativos): puede almacenar valores numéricos desde 9223372036854775807 hasta -9223372036854775808.

**TINYINT**

TINYINT sería bueno usar para números que serán relativamente pequeños. Un buen ejemplo de algo que usaría un TINYINT es el identificador de nivel de usuario (0 - usuario inactivo, 1 - usuario activo, 9 - administrador).

**sin signo:** puede almacenar valores numéricos desde 0 hasta 255

**con signo:**puede almacenar valores numéricos desde -128 hasta 127

**FLOAT**

Se usa para almacenar números de coma flotante (números que necesitan tener decimales). Una columna de ejemplo para esto sería como un item\_cost.

**TEXT**

Se usa para almacenar una gran cantidad de texto, como una descripción, mensaje o comentario. Usa esto para cualquier texto que VARCHAR () sea demasiado pequeño para manejar.

**DATE**

Se usa para almacenar una fecha y hora en el formato AAAA-MM-DD hh: mm: ss

* Nombres -> Varchar(longitud)
* Direcciones -> Varchar(longitud)
* Teléfonos -> Varchar o Int, aunque en realidad setear un teléfono como número es un tanto innecesario ya que nunca usarás una operación matemática (me refiero a sumar, restar, etc) con un teléfono.
* Correos -> Varchar(longitud)
* Fechas -> Date o Datetime, dependiendo lo que requieras y la versión de SQL que estés utilizando.
* Imágenes -> Varchar(longitud)
* Números enteros -> Int o BigInt dependiendo el límite del número a ingresar.
* Números con decimal -> Decimal