

¿Porqué es necesaria la Normalización de un modelo relacional?

Una base de datos tiene que ser diseñada antes de que pueda ser creada y usada. El diseño debe ajustarse a estándares que permitan ahorro de memoria, acceso rápido, fácil mantenimiento, portabilidad, facilidad de futuros mejoramientos, buen desempeño y eficiencia de costos, entre otros. El diseño lógico final de una base de datos debe ser tal que equilibre un desempeño óptimo junto con la integridad de la información. Esto puede ser logrado a través de un proceso conocido como Normalización. La base de datos debe estar en un estado de "Forma completamente normalizada".

DEFINICIÓN DE NORMALIZACION

Normalización es una serie de reglas que involucra análisis y transformación de las estructuras de los datos en relaciones que exhiban propiedades únicas de consistencia, mínima redundancia y máxima estabilidad.

La necesidad para normalizar puede ser mejor comprendida al mencionar las distintas anomalías o desventajas de los datos NO NORMALIZADOS. Consideremos la tabla en la figura 3. La tabla contiene todos los detalles de los empleados de una compañía, y los detalles del Departamento al que pertenecen.

EmpleadosCompañía (#Empleado, NombreEmpleado, Dir_empleado,#numDept, descDept)

Figura 3

A primera vista, parece conveniente almacenar todos los detalles en una sola tabla. Pero ciertas anomalías se pueden manifestar durante la inserción, actualización y borrado de datos. La normalización provee un método de remover todas estas indeseables anomalías haciendo la base de datos mas confiable y estable.

Anomalía de inserción (INSERT)

Suponga que un nuevo Departamento ha sido creado, el cual no tiene empleados todavía, por lo tanto, en nuestra tabla original, los datos correspondientes al empleado estarían vacíos (nulos), y solo tendríamos la información del Departamento: Columnas "numDept" y "descDept".

Anomalía de Actualización (UPDATE)

Suponga que el número del Departamento de "Sistemas" ha sido cambiado a AB108. Esto involucra tener que cambiar el numero del departamento para todos los empleados que pertenezcan al departamento de "Sistemas", lo cual representa tiempo y recursos de sistema adicionales.

Anomalía de borrado (DELETE)

Si todos los empleados en el Departamento de "Finanzas" abandonan la compañía, todos los registros de estos tendrían que ser borrados. Hecho así, los detalles del Departamento "Finanzas" se perderían. Los datos en la tabla entonces no representan una información correcta sobre el estado de la compañía, y por lo tanto se pierde la integridad de los datos.

PROPIEDADES DE UNA BASE DE DATOS DESPUÉS DE LA NORMALIZACION

Una base de datos normalizada debe representar las siguientes propiedades:

- Los requerimientos para almacenamiento de datos se minimizan, dado que el proceso de normalización sistemáticamente elimina la duplicación de los datos.
- Desde que los datos son almacenados en el mínimo número de lugares, las posibilidades de inconsistencias en la información son reducidas al mínimo.
- Las estructuras normalizadas son óptimas para efectuar actualizaciones de los datos. Dado que los datos existen en el mínimo número de lugares, una operación de actualización (UPDATE) necesitará acceder a una mínima cantidad de datos.

PROCEDIMIENTOS DE NORMALIZACION

El proceso de normalización involucra básicamente tres pasos. Después de cada paso, la base de datos se convierte en formas llamadas "formas normales". Generalmente, la "tercera forma normal" es el estado que debe alcanzar una base de datos para que se diga que está totalmente normalizada. La cuarta y la quinta forma normal también existen, pero no son usadas en el diseño de una base de datos.

PROPIEDADES DE UNA RELACIÓN (O *TABLA RELACIONAL*)

Un tabla debe satisfacer ciertos criterios previos antes de calificar para convertirse en una relación.

No duplicados

No debe haber nunca dos columnas o filas totalmente idénticas. Si dos filas son totalmente idénticas, entonces hacen falta algunos atributos que las haga diferentes y distinguibles. Ejemplo: Dos registros de discos compactos en una tienda serían idénticos si son dos copias del último álbum de Shakira, si no fuera porque cada disco compacto tiene un número código que los hace diferentes.

Clave Única

Cada registro tiene que tener una clave única que lo identifique. Cualquier atributo puede ser una clave, pero en lo posible trataremos de elegir como clave única al atributo que tenga una longitud menor y fija, como por ejemplo un número de ID. Si un atributo es insuficiente para identificar un registro de manera única, entonces más de un atributo puede conformar la clave única. En tal caso, el número de atributos que conformen una llave debe ser el mínimo necesario y suficiente.

Insignificancia del orden

La secuencia en la cual los atributos son escritos no debe importar. Podemos escribir el ID del empleado de primero, o el nombre y el apellido de primero, y esto no afectará las relaciones que establezcamos con otras tablas. Por otro lado, los registros deben ser totalmente independiente de su secuencia o posición en la base de datos (dependencia posicional). Esto significa que si intentamos identificar un registro por su posición dentro de la tabla, estaremos creando una llave inválida.

Forma no-normalizada

Los datos, en su forma elemental, no están normalizados. Por lo tanto, lo primero con lo que debemos comenzar es con los datos elementales o básicos que conformarán el diccionario de datos. El diccionario de datos es creado a partir de los documentos o diagramas de flujo de la compañía. Se deben listar los elementos uno debajo del otro. Así, obtendremos la forma no-normalizada para el ejercicio de ARD (Análisis Relacional de Datos), con el cual deberemos obtener al final distintos grupos de elementos. Mas tarde, dichos grupos se combinarán con los grupos de otros documentos al cual también se les ha hecho el análisis ARD, y se establecerán relaciones entre ellos.

Normalización de Bases de Datos (con un ejemplo)

Uno de los factores mas importantes en la creación de páginas web dinámicas es el diseño de las Bases de Datos (BD). Si tus tablas no estan correctamente diseñadas, te pueden causar un montón de dolores de cabeza cuando tengas de realizar complicadísimas llamadas SQL en el código PHP para extraer los datos que necesitas. Si conoces como establecer las relaciones entre los datos y la normalización de estos, estarás preparado para comenzar a desarrollar tu aplicación en PHP.

Si trabajas con MySQL o con Oracle, debes conocer los métodos de normalización del diseño de las tablas en tu sistema de BD relacional. Estos métodos pueden ayudarte a hacer tu código PHP mas fácil de comprender, ampliar, y en determinados casos, incluso hacer tu aplicación mas rápida.

Básicamente, las reglas de Normalización están encaminadas a eliminar redundancias e inconsistencias de dependencia en el diseño de las tablas. Más tarde explicaré lo que esto significa mientras vemos los cinco pasos progresivos para normalizar, tienes que tener en cuenta que debes crear una BD funcional y eficiente. También detallaré los tipos de relaciones que tu estructura de datos puede tener.

Digamos que queremos crear una tabla con la información de usuarios, y los datos a guardar son el nombre, la empresa, la dirección de la empresa y algún e-mail, o bien URL si las tienen. En principio comenzarías definiendo la estructura de una tabla como esta:

Relación No Normalizada (“Formalización CERO “)

usuarios				
<u>userid</u>	nombre	empresa	dirección_empresa	url
1	Joe	ABC	1 Work Lane	abc.com xyz.com
2	Jill	XYZ	1 Job Street	abc.com xyz.com
3	Jane	XYZ	1 Job Street	abc.com

Diríamos que la anterior tabla está No Normalizada (nivel de Formalización “Cero”) porque ninguna de nuestras reglas de normalización ha sido aplicada. Observa que en el campo 'url' se tienen asociados varios valores

Diseñar la tabla con dos campos url1 y url2, no es una solución: ¿Qué haremos cuando en nuestra aplicación necesitemos una tercera url ? ¿ Quieres tener que añadir otro campo/columna a tu tabla y tener que reprogramar toda la entrada de datos de tu código PHP ? Obviamente no, tu quieres crear un sistema funcional que pueda crecer y adaptarse fácilmente a los nuevos requisitos. Echemos un vistazo a las reglas del Primer Nivel de Formalización-Normalización, y las aplicaremos a nuestra tabla.

Primer nivel de Formalización/Normalización. (F/N)

¿Ves que aparece un grupo repetitivo en el campo url para una misma fila?. Si duplicamos la fila, cambiando el valor de url en cada copia,, como la clave es '**userid**', se viola la regla de unicidad de valor del campo clave (dos tuplas tendrían el mismo valor de campo clave).

Para que una relación esté en 1ª Forma Normal, hay que eliminar los grupos repetitivos de la tablas individuales. Eso se consigue de alguna de estas dos maneras:

1. Crear una tabla separada por cada grupo de datos relacionados, identificar cada grupo de datos relacionados con una clave primaria.
2. Considerar una única tabla, eligiendo adecuadamente como clave primaria una combinación de campos que eliminen los grupos repetitivos, es decir, eligiendo como clave primaria una combinación de campos que se único para cada tupla (en el peor de los casos, serían todos los campos de esa única relación). En este caso, nos encontraríamos con la siguiente tabla, cuya clave estaría formada por la composición de los campos "**userid,url**":

Usuarios-urls-Empresas				
userid	nombre	empresa	dirección_empresa	url
1	Joe	ABC	1 Work Lane	abc.com
1	Joe	ABC	1 Work Lane	xyz.com
2	Jill	XYZ	1 Job Street	abc.com
2	Jill	XYZ	1 Job Street	xyz.com
3	Jane	XYZ	1 Job Street	abc.com

Ahora diremos que nuestra tabla está en el primer nivel de F/N. Hemos solucionado el problema de los grupos repetitivos (en este caso, el problema de los múltiples url). Sin embargo vemos otros problemas....Cada vez que introducimos un nuevo registro en la **tabla usuarios**, tenemos que duplicar el nombre de la empresa y del usuario. No sólo nuestra BD crecerá muchísimo, sino que será muy fácil que la BD se corrompa si escribimos mal alguno de los datos redundantes. Aplicaremos pues el segundo nivel de F/N:

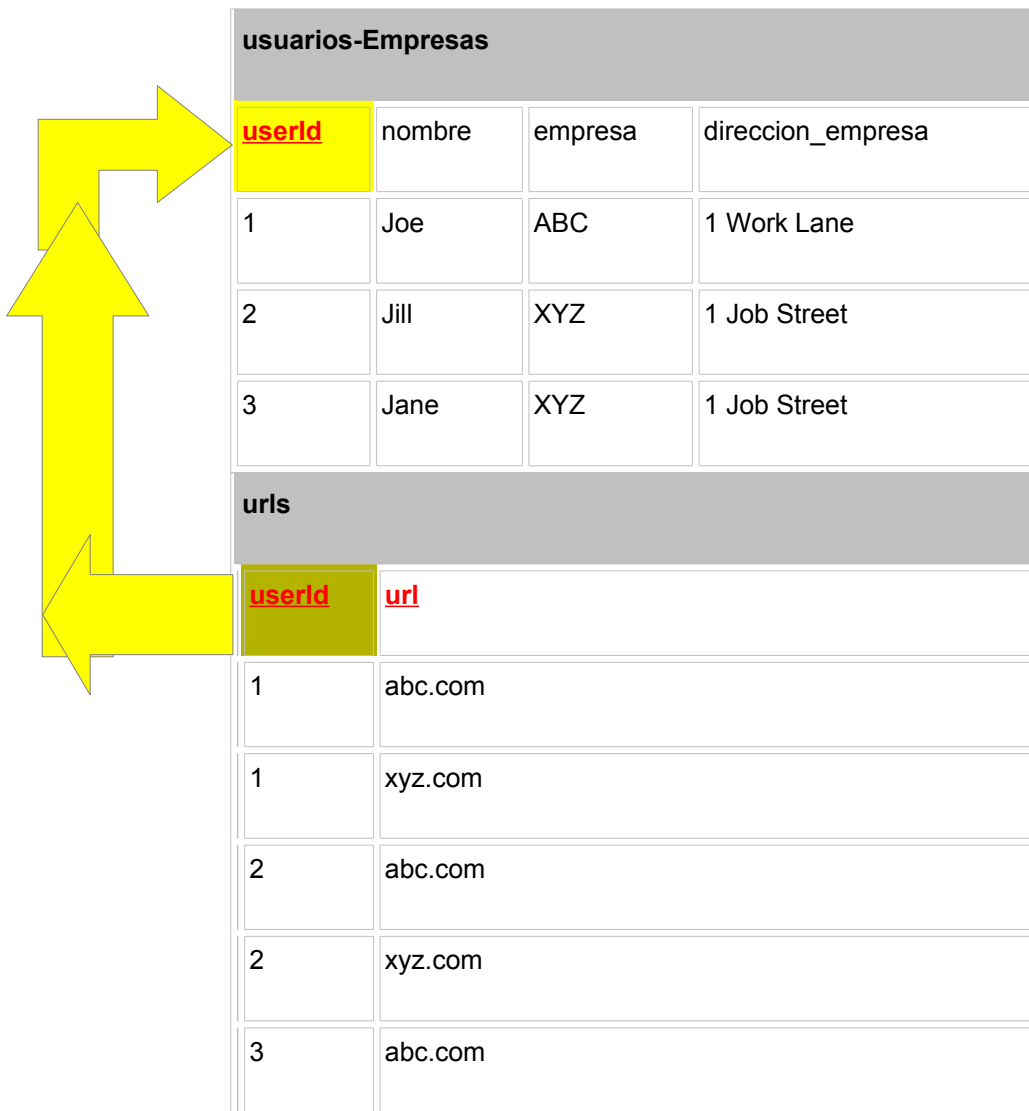
Segundo nivel de F/N

¿Ves que se está repitiendo innecesariamente el nombre del empleado, el nombre y la dirección de la empresa en las filas que corresponden a un mismo empleado?. Eso se debe a que para un mismo código de empleado le corresponde una misma Empresa; a su vez, para una misma empresa, se tiene asociada siempre la misma dirección.

Para que una relación esté en 2ª Forma Normal, deben localizarse los campos que dependan de parte de la clave primaria y “sacarlos” a otra nueva tabla, relacionando ambas tablas mediante integridad referencial. En concreto los pasos a seguir son:

1. Localizar campos que dependen de parte de la clave primaria y a su vez los campos que dependen de esos campos.
2. Crear tablas separadas para aquellos grupos de campos (tanto los que dependen de parte de la clave, como los que a su vez dependen de estos)
3. Relacionar estas tablas mediante una clave externa.

Hemos “sacado” el `userid` y nombre del usuario, el nombre y la dirección de la empresa en una nueva tabla. En la tabla original se mantienen los campos de la clave primaria de la tabla original, **`userid`** y **`url`**. De esta forma podemos añadir más en el futuro mas url si tener que duplicar los demás datos. Se usa **`userid`** para relacionar estos campos:



usuarios-Empresas			
<u>userId</u>	nombre	empresa	direccion_empresa
1	Joe	ABC	1 Work Lane
2	Jill	XYZ	1 Job Street
3	Jane	XYZ	1 Job Street
urls			
<u>userId</u>	<u>url</u>		
1	abc.com		
1	xyz.com		
2	abc.com		
2	xyz.com		
3	abc.com		

Vale, hemos creado tablas separadas y la clave primaria en la **tabla usuarios**, **userId**, esta relacionada ahora con la **clave externa** en la **tabla urls**, **userId**. Esto esta mejor. ¿ Pero que ocurre cuando queremos añadir otro empleado a la empresa ABC ? ¿ o 200 empleados ? Ahora tenemos el nombre de la empresa y su dirección duplicándose, otra situación que puede inducirnos a introducir errores en nuestros datos. Así que tendremos que aplicar el tercer nivel de F/N:

Tercer nivel de F/N

¿Ves que en la tabla de usuarios se repiten innecesariamente los datos de las empresas?. Eso se debe a que los datos de la empresa dependen solo del nombre de la empresa, no del usuario.

Para que una relación esté en 3ª Forma Normal, hay que localizar los campos que dependen de un campo que no es la clave (pueden depender de parte de la clave, puesto que la tabla ya está en 2ª Forma Normal) y llevarlos a una nueva tabla. En concreto los pasos son:

1. Localizar los campos que dependen de algún(os) campos que no es la clave principal
2. Crear tablas separadas para esos campos
3. La tabla original y la nueva tabla se relación mediante integridad referencia con el campo del que dependen los atributos desplazados a la nueva tabla.

Nuestro nombre de empresa y su dirección no tienen nada que ver con el campo userId, así que tienen que tener su propio empresaId:

usuarios		
userId	nombre	relEmpresaId
1	Joe	1
2	Jill	2
3	Jane	2
empresas		
emprId	empresa	direccion_empresa
1	ABC	1 Work Lane
2	XYZ	1 Job Street
urls		
	userId	url
	1	abc.com
	1	xyz.com
	2	abc.com
	2	xyz.com
	3	abc.com

Ahora tenemos la clave primaria **emprId** en la **tabla empresas** relacionada con la clave externa **recEmpresaId** en la **tabla usuarios**, y podemos añadir 200 usuarios mientras que sólo tenemos que insertar el nombre 'ABC' una vez. Nuestras tablas de usuarios y urls pueden crecer todo lo que quieran sin duplicación ni corrupción de datos. La mayoría de los desarrolladores dicen que el tercer nivel de F/N es suficiente, que nuestro esquema de datos puede manejar fácilmente los datos obtenidos de una cualquier empresa en su totalidad, y en la mayoría de los casos esto será cierto.

Pero echemos un vistazo a nuestro campo urls - ¿ Ves duplicación de datos ? Esto es perfectamente aceptable si la entrada de datos de este campo es solicitada al usuario en nuestra aplicación para que teclee libremente su url, y por lo tanto es sólo una coincidencia que Joe y Jill teclearon la misma url. ¿ Pero que pasa si en lugar de entrada libre de texto usáramos un menú desplegable con 20 o incluso más urls predefinidas ? Entonces tendríamos que llevar nuestro diseño de BD al siguiente nivel de F/N, el cuarto, muchos desarrolladores lo pasan por alto porque depende mucho de un tipo muy específico de relación, la relación 'varios-con-varios', la cual aún no hemos encontrado en nuestra aplicación.

Relaciones entre los Datos

Antes de definir el cuarto nivel de F/N, veremos tres tipos de relaciones entre los datos: uno-a-uno, uno-con-varios y varios-con-varios.

uno-a-uno

Mira la **tabla usuarios** en el Primer Nivel de F/N del ejemplo de arriba. Por un momento imaginemos que ponemos el campo url en una tabla separada, y cada vez que introducimos un registro en la **tabla usuarios** también introducimos una sola fila en la **tabla urls**. Entonces tendríamos una relación uno-a-uno: cada fila en la tabla usuarios tendría exactamente una fila correspondiente en la tabla urls. Para los propósitos de nuestra aplicación no sería útil la normalización.

uno-a-varios

Ahora mira las tablas en el ejemplo del Segundo Nivel de F/N. Nuestras tablas permiten a un sólo usuario tener asociadas varias urls (eliminando redundancias). Esta es una relación uno-con-varios, el tipo de relación más común.

En el ejemplo del Tercer Nivel de F/N, tenemos tablas que permiten a una misma empresa tener varios empleados (eliminando redundancias).

varios-a-varios

La relación varios-con-varios, sin embargo, es ligeramente más compleja. Observa en nuestro ejemplo del Tercer Nivel de F/N que tenemos en la **Tabla urls** a un usuario relacionado con varias urls y una misma url puede estar asociada a diferentes usuarios. Se está produciendo una redundancia, puesto que aparece varias veces repetida la misma url.

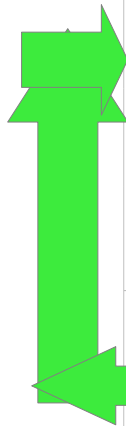
Como dijimos, vamos a cambiar la estructura para permitir que varios usuarios estén relacionados con varias urls previamente definidas y así tendremos una relación varios-con-varios, puesto que una misma url puede estar asociada a varios usuarios.

Cuarto Nivel de F/N

1. En las relaciones varios-con-varios, entidades independientes no pueden ser almacenadas en la misma tabla.

Ya que sólo se aplica a las relaciones varios-con-varios, la mayoría de los desarrolladores pueden ignorar esta regla de forma correcta. Pero es muy útil en ciertas situaciones, tal como esta. Hemos optimizado nuestra **tabla urls** eliminado duplicados y hemos puesto las relaciones en su propia tabla.

usuarios		
userId	nombre	relEmpresald
1	Joe	1
2	Jill	2
3	Jane	2
empresas		
emprld	empresa	direccion_empresa
1	ABC	1 Work Lane
2	XYZ	1 Job Street
urls		
urlld	url	
1	abc.com	
2	xyz.com	
url_relations		
relationId	relatedUrlId	relatedUserId
1	1	1
2	1	2
3	2	1
4	2	2



5	3	1
---	---	---

Para disminuir la duplicación de los datos (este proceso nos llevará al Cuarto Nivel de F/N), hemos creado una tabla que sólo tiene claves externas y primarias url_relations. Hemos sido capaces de eliminar la entradas duplicadas en la tabla urls creando la **tabla url_relations**. Ahora podemos expresar fielmente la relación que ambos Joe and Jill tienen entre cada uno de ellos, y entre ambos, las urls. Así que veamos exactamente que es lo que el Cuarto Nivel de F/N. supone:

Os voy a poner un ejemplo práctico, ahora podemos seleccionar todas las urls de Joe realizando la siguiente instrucción SQL:

```
SELECT nombre, url FROM usuarios, urls, url_relations WHERE  
url_relations.relatedUserId = 1 AND usuarios.userId = 1 AND urls.urlId =  
url_relations.relatedUrlId
```

Y si queremos recorrer todas las urls de cada uno de los usuarios, haríamos algo así:

```
SELECT nombre, url FROM usuarios, urls, url_relations WHERE usuarios.userId =  
url_relations.relatedUserId AND urls.urlId = url_relations.relatedUrlId
```

Quinto Nivel de F/N

Existe otro nivel de normalización que se aplica a veces, pero es de hecho algo esotérico y en la mayoría de los casos no es necesario para obtener la mejor funcionalidad de nuestra estructura de datos o aplicación. Su principio sugiere:

1. La tabla original debe ser reconstruida desde las tablas resultantes en las cuales a sido troceada.

Los beneficios de aplicar esta regla aseguran que no has creado ninguna columna extraña en tus tablas y que la estructura de las tablas que has creado sea del tamaño justo que tiene que ser. Es una buena práctica aplicar esta regla, pero a no ser que estés tratando con una extensa estructura de datos probablemente no la necesitarás.

--Barry ,Traducido por. J.M.Font (jmfg21@wanadoo.es)