Fundamentos de Base de Datos I

Ing. Francisco José Santana V. MTi

Capítulo IV

IV- Normalización

BREVE RESEÑA HISTORICA

El creador del proceso de normalización de Bases de Datos fue Edgar. F. Codd quien entre 1968 y 1970 desarrollo los conceptos básicos del "modelo relacional" de Bases de Datos. Posteriormente entre 1970 y 1973 E. Codd escribió varios artículos referidos a la leyes de Armstrong sobre la álgebra relacional. Todo este proceso de investigación finalizó en el desarrollo del Proyecto "Ingres", considerado una de las primeras bases de datos basada en la concepción del modelo relacional.

Para entonces E. Codd había formulado las 3 Formas Normales (FN) Básicas (objeto de nuestro estudio), a las cuales se sumaron con posterioridad 2 Formas Normales más postuladas por Fagin y una redefinición de la 3FN a la cual se llamó la Forma Normal de Boyce-Codd.

Qué es la normalización

La **normalización** es el proceso mediante el cual se transforman datos complejos a un conjunto de estructuras de datos más pequeñas, que además de ser más simples y más estables, son más fáciles de mantener. También se puede entender la **normalización** como una serie de reglas que sirven para ayudar a los diseñadores de bases de datos a desarrollar un esquema que minimice los problemas de lógica. Cada regla está basada en la que le antecede.

La normalización se adoptó porque el viejo estilo de poner todos los datos en un solo lugar, como un archivo o una tabla de la base de datos, era ineficiente y conducía a errores de lógica cuando se trataban de manipular los datos. La normalización también hace las cosas fáciles de entender. Los seres humanos tenemos la tendencia de simplificar las cosas al máximo. Lo hacemos con casi todo, desde los animales hasta con los automóviles. Vemos una imagen de gran tamaño y la hacemos más simple agrupando cosas similares juntas. Las guías que la normalización provee el marco de referencia para simplificar una estructura de datos compleja.

Otra ventaja de la normalización de base de datos es el consumo de espacio. Una base de datos normalizada ocupa menos espacio en disco que una no normalizada. Hay menos repetición de datos, lo que tiene como consecuencia menor uso de espacio en disco.

El proceso de normalización tiene un nombre y una serie de reglas para cada fase. Esto puede parecer un poco confuso al principio, pero poco a poco se va entendiendo el proceso, así como las razones para hacerlo de esta manera. Tambien puede decirse **Normalización:** Es un proceso de organizar los elementos de datos de una o varias estructuras, para:

- Reducir Redundancia
- Identificar las dependencias
- Simplificar las estructuras

Entidades Normalizadas:

Una relación totalmente normalizada consiste en una clave primaria que identifica alguna entidad de datos, junto a un grupo de atributos que la describen.

Reglas de Normalización Resumidas

Primera Forma Normal (1NF):

Una entidad está en la primera forma normal (1FN) si todos los atributos son simples, es decir, la relación no contiene grupos repetitivos.

Un Grupo repetitivo, no es más que un atributo o grupo de atributos que pueden tomas más de un valor u ocurrecia dentro de cada tupla o tipo entidad.

Segunda Forma Normal (2NF):

Una entidad está en la (2FN) si está en la 1NF y todo atributo no clave depende funcional y totalmente de la clave primeria.

Tercera Forma Normal (3NF):

Una entidad está en la (3FN) si está en 2NF y cada atributo no clave es funcionalmente dependiente de la clave primaria e independiente de cualquier otro atributo no clave.

Caso Práctico :

Formalización CERO o Forma Normal Cero (**0FN**)

Usuarios				
Nombre	Empresa	Dir_empresa	Email1	Email2
Joe	ABC	1 Work Lane	j@abc.com	j@xyz.com
Hill	XYZ	1 Job Street	h@abc.com	h@xyz.com

Diríamos que la anterior tabla esta en nivel de Formalizacion Cero porque ninguna de nuestras reglas de normalización ha sido aplicada. Observa los campos Email1 y Email2 -- ¿Qué haremos cuando en nuestra aplicación necesitemos un tercer Email? ¿Quieres tener que añadir otro campo/columna a tu tabla y tener que reprogramar toda la entrada de datos de tu código? Obviamente no, tú 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: Primera Forma Normal (1FN)

1. Eliminar los grupos repetitivos de las tablas individuales.
2. Crear una tabla separada por cada grupo de datos relacionados.
3. Identificar cada grupo de datos relacionados con una clave primaria.

¿Ves que estamos rompiendo la primera regla cuando repetimos los campos Email1 y Email2?

¿ Y que pasa con la tercera regla, la clave primaria? La regla tres básicamente significa que tenemos que poner un campo tipo contador autoincrementable para cada registro. De otra forma, ¿Qué pasaría si tuviéramos dos usuarios llamados Joe y queremos diferenciarlos. Una vez que aplicáramos el primer nivel de F/N nos encontraríamos con la siguiente tabla:

Usuarios				
UserId	Nombre	Empresa	Dir_empresa	Email
1	Joe	ABC	1 Work Lane	j@abc.com
1	Joe	ABC	1 Work Lane	j@xyz.com
2	Hill	XYZ	1 Job Street	h@abc.com
2	Hill	XYZ	1 Job Street	h@xyz.com

Ahora diremos que nuestra tabla está en 1FN. Hemos solucionado el problema de la limitación del campo email. Pero 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 (2FN):

Segunda nivel: Segunda Forma Normal (2FN)

Crear tablas separadas para aquellos grupos de datos que se aplican a varios registros.

4. Relacionar estas tablas mediante una clave externa.

   Hemos separado el campo Email en otra tabla, de forma que podemos añadir más en el futuro si tener que duplicar los demás datos. También vamos a usar nuestra clave primaria para relacionar estos campos:

Usuarios			
userId	nombre	empresa	Dir_empresa
1	Joe	ABC	1 Work Lane
2	Hill	XYZ	1 Job Street

Emails		
EmailId	RelUserId	Email
1	1	j@abc.com
2	1	j@xyz.com
3	2	h@abc.com
4	2	h@xyz.com

Hemos creado tablas separadas y la clave primaria en la **tabla usuarios**, userId, esta relacionada ahora con la clave externa en la **tabla Emails**, relUserId. 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, (3FN):

Tercer nivel: Tercera Forma Normal (3NF).

5. Eliminar aquellos campos que no dependan de la clave primaria.

   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	Hill	2

Empresas		
emprId	empresa	Dir_empresa
1	ABC	1 Work Lane
2	XYZ	1 Job Street

Emails		
EmailId	RelUserId	Email
1	1	j@abc.com
2	1	j@xyz.com
3	2	h@abc.com
4	2	h@xyz.com

Ahora tenemos la clave primaria emprId en la **tabla empresas** relacionadas 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 Emails 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 Emails - ¿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 Email, y por lo tanto es sólo una coincidencia que Joe y Hill teclearon la misma Email. ¿Pero que pasa si en lugar de entrada libre de texto usáramos un menú desplegable con 20 o incluso más Emails 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 (4FN), veremos tres tipos de relaciones entre los datos: uno-a-uno, uno-con-varios y varios-con-varios. Mira la **tabla usuarios** en el Primer Nivel de F/N (1FN) del ejemplo de arriba. Por un momento imaginemos que ponemos el campo Email en una tabla separada, y cada vez que introducimos un registro en la **tabla usuarios** también introducimos una sola fila en la **tabla Emails**. Entonces tendríamos una relación uno-a-uno: cada fila en la tabla usuarios tendría exactamente una fila correspondiente en la **tabla Emails**. Para los propósitos de nuestra aplicación no sería útil la normalización.

Ahora mira las tablas en el ejemplo del Segundo Nivel de F/N. Nuestras tablas permiten a un sólo usuario tener asociadas varias Emails. Esta es una relación uno-con-varios, el tipo de relación más común, y hasta que se nos presentó el dilema del Tercer Nivel de F/N. la única clase de relación que necesitamos.

La relación varios-con-varios, sin embargo, es ligeramente más compleja. Observa en nuestro ejemplo del Tercer Nivel de F/N que tenemos a un usuario relacionado con varias Emails. Como dijimos, vamos a cambiar la estructura para permitir que varios usuarios estén relacionados con varias Emails y así tendremos una relación varios-con-varios. Veamos como quedarían nuestras tablas antes de seguir con este planteamiento:

Usuarios		
userId	Nombre	relEmpresaId
1	Joe	1
2	Hill	2

Empresas		
emprId	empresa	Dir_empresa
1	ABC	1 Work Lane
2	XYZ	1 Job Street

Emails	
EmailId	Email
1	j@abc.com
2	h@xyz.com
3	j@xyz.com
4	h@abc.com

Email_relations		
relationId	RelatedEmailId	relatedUserId
1	1	1
2	3	2
3	2	1
4	4	2

Para disminuir la duplicación de los datos (este proceso nos llevará al Cuarto Nivel de F/N (4FN), hemos creado una tabla que sólo tiene claves externas y primarias Email_relations. Hemos sido capaces de remover las entradas duplicadas en la tabla Emails creando la tabla Email_relations. Ahora podemos expresar fielmente la relación que ambos Joe y Hill tienen entre cada uno de ellos, y entre ambos, las Emails. Así que veamos exactamente que es lo que el Cuarto Nivel de F/N (4FN). Supone:

Cuarto Nivel: Cuarta Forma Normal (FN4)

6. 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 Emails** eliminado duplicados y hemos puesto las relaciones en su propia tabla.

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

SELECT nombre, Email FROM usuarios, Emails, Email_relations WHERE Email_relations.relatedUserId = 1 AND usuarios.userId = 1 AND Emails.EmailId = Email_relations.relatedEmailId

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

SELECT nombre, Email FROM usuarios, Emails, Email_relations WHERE usuarios.userId = Email_relations.relatedUserId AND Emails.EmailId = Email_relations.relatedEmailId

Quinto Nivel: Quinta Forma Normal (FN5)

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 dividida.

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.

Consejos Útiles para el Diseño de Bases de Datos

Al momento de trabajar en el Diseño e Implementación de una Base de Datos deben tomarse en cuenta diferentes consejos (Tips) que permiten un diseño más apropiado, funcional y estructurado de nuestra base de datos y por ente un mejor desempeño.

A continuación listamos algunos de ellos:

1. Los registros son gratis, los campos nuevos son caros

Las bases de datos se deben diseñar de forma que lo único que haya que hacer conforme pasa el tiempo sea agregar nuevos registros. Las tablas de las bases de datos están diseñadas para alojar grandes cantidades de registros. No obstante, si se da cuenta de que debe agregar más campos, puede que tenga un problema de diseño.

Esto suele suceder con expertos en hojas de cálculo, que suelen diseñar bases de datos del mismo modo en que acostumbran a diseñar hojas de cálculo. El diseño de campos sujetos a limitaciones temporales (como, por ejemplo, Año, Trimestre, Producto y Vendedor) hace que sea necesario agregar nuevos campos en el futuro. Pero el diseño correcto consiste en transponer la información y tener los datos sujetos a limitaciones temporales en un solo campo, de forma que se puedan agregar más registros. Por ejemplo, en lugar de crear un campo distinto para cada año, cree un campo Año y escriba el valor de año de cada registro en ese campo.

La razón por la que resulta problemático agregar campos adicionales es el impacto que tiene realizar cambios estructurales en las tablas sobre otras partes de la aplicación. Cuando se agregan campos a una tabla, los objetos y el código que dependen de la tabla también deben actualizarse. Por ejemplo, las consultas deben abarcar los campos extra, los formularios deben mostrarlos, los informes deben incluirlos, etc. Sin embargo, si los datos estuvieran normalizados, los objetos existentes recuperarían automáticamente los nuevos datos y los calcularían o mostrarían correctamente. Las consultas son especialmente eficaces, ya que permiten centrarse en el campo Año para mostrar resúmenes por año, independientemente de qué años haya en la tabla.

No obstante, la normalización de datos no significa que no se puedan mostrar o utilizar datos con campos sujetos o dependientes de limitaciones temporales. Los desarrolladores que deban mostrar tal información pueden hacerlo mediante consultas de tabla de referencias cruzadas. Si desconoce el uso de estas consultas, es recomendable que aprenda a utilizarlas. No son lo mismo que las tablas (en concreto, los resultados de una consulta de tabla de referencias cruzadas no se pueden editar), pero sí pueden usarse para mostrar información en una hoja de datos (de hasta 255 campos). Si desea utilizarla en informes, resulta más complicado, ya que el informe tendrá que ajustarse a los nombres de campo adicionales o que cambien. Esa es la razón de que, en la mayoría de los informes, los datos se muestren como agrupaciones aparte dentro del informe, en lugar de como columnas aparte. En los casos en los que no tenga otra elección, tendrá que invertir tiempo para poder hacer esto, pero seguramente todas las partes implicadas entenderán la implicación que estas decisiones tienen sobre los recursos adicionales con el transcurso del tiempo.

En definitiva, este es el motivo de que los registros adicionales sean gratis (la gran ventaja de las bases de datos) y los campos adicionales sean tan caros. Las bases de datos se pueden ajustar a enormes cantidades de cambios, si están diseñadas correctamente.

2. Campos para el control de Modificación en cada registro:

Se recomienda que cada Tabla de la base de Datos incluya dentro de sus campos operaciones, Fecha de Creación del Registro, Fecha de Modificación y el nombre de usuario que ha realizado dichos cambios. Esto va a permitir que cada vez que se agregue un registro a la base de datos o se modifique, la aplicación pueda registrar que usuario y en qué fecha está realizando dicho cambio se utilizando la sentencia Delete u Update.

3. Cuándo deben duplicarse los datos?

A veces, es necesario deshacer la normalización de los datos para conservar información que puede cambiar con el paso del tiempo.

En el sencillo ejemplo anterior de una factura vinculada a un cliente mediante un número de Id. de cliente, puede que deseemos conservar la dirección del cliente en el momento en que se emitió la factura (no en el momento de su creación, ya que la información del cliente puede cambiar entre los dos eventos). Si no conservamos la dirección en el momento en que se emitió la factura y tuvimos que actualizar la información del cliente después de dicha emisión, tal vez no podamos confirmar la dirección exacta a la que se envió la factura. Esto podría causar un enorme problema empresarial. Por supuesto, hay ciertos tipos de información, como el número de teléfono del cliente, que no es imprescindible conservar. Por lo tanto, se debe determinar de forma selectiva qué datos hay que duplicar.

Otro ejemplo en el que los datos deben duplicarse es al rellenar los artículos de línea de una factura. A menudo, se utiliza una lista de precios para elegir los artículos que ha pedido el cliente. Se podría simplemente almacenar el Id. de la lista de precios para que apunte a la lista de precios con la descripción, el precio y otros detalles del producto. Sin embargo, las descripciones y precios de los productos cambian al paso del tiempo. Si no copia los datos de la lista de precios en la tabla de artículos de línea, no podrá volver a imprimir con precisión la factura original en el futuro, lo que puede constituir un enorme problema si aún no ha recibido el importe de la factura.

Por lo tanto, aunque la normalización resulta apropiada para guardar los mismos datos en un sitio y, además, simplifica la edición, hay situaciones en las que no se desea disponer de esas ventajas. Si necesita una instantánea de los datos por motivos de historial, es muy importante que lo implemente en el diseño de la base de datos desde el principio. Si no es así, una vez que los datos se sobrescriban, no se podrán recuperar.

4. La estructura de la Base de Datos debe ser Estática:

Al momento de diseñar, implementar y Desarrollar una base de Datos, es importante capturar todos los elementos importantes del Minimundo o Escenario que se pretende modelar. Es por ello que todos estos aspectos deben recogerse al momento del diseño. En caso de ser necesario implementar cambios en la estructura de la base de datos, es necesario siempre hacerlo en una copia de la base de datos que no esté accedida por los usuarios, interactuando con alguna otra aplicación o que en términos informáticos no se encuentre en producción. Dichos cambios deben ser evaluados por el DBA, los diseñadores o cualquier usuario al que le afectarían dichos cambios.

5. Uso de un campo sin significado en el campo de clave

Por motivos de eficacia, cada tabla debe tener un campo de clave. El campo de clave define la exclusividad en la tabla; los índices lo utilizan en los otros campos para mejorar el rendimiento de las búsquedas. Por ejemplo, la tabla de clientes podría tener un campo CustomerID que definiera un número exclusivo para cada cliente. En esta discusión, se habla de tablas que tienen varios campos y no de simples consultas en tablas sencillas, como, por ejemplo, una lista de países.

En general, un campo de clave debe tener las siguientes características:

- **Debe ser un solo campo:** Se pueden definir varios campos como campos de clave de una tabla, pero es preferible que sólo haya un único campo. En primer lugar, si son necesarios varios campos para definir la exclusividad, se precisa de más espacio para almacenar la clave. En segundo lugar, los índices adicionales de la tabla también tienen que usar la combinación de los campos de clave, lo que ocupa más espacio que si se tratara de un único campo. Por último, la identificación de registros en la tabla requiere abarcar una combinación de campos. Es mucho mejor tener un campo CustomerID que una combinación de otros campos para definir un cliente.

- **Debe ser numérico:** la mayoría de los Manejadores de Bases de Datos ofrecen un tipo de campo **Autonumérico** que es un **Entero largo**, lo que resulta ideal para los campos de clave. Estos valores se convierten automáticamente en exclusivos para cada campo y, además, admiten la entrada de datos por parte de varios usuarios.

- **No debe cambiar con el tiempo:** Los campos de clave no deben cambiar con el tiempo. Al igual que un número de la seguridad social, una vez identificados, nunca deben cambiar. Un campo de clave que cambie dificulta en gran medida el uso de datos históricos, ya que los vínculos se rompen.

- **Debe ser sin significado:** Para garantizar que un campo de clave no cambie con el tiempo, éste no debe tener ningún significado. Los valores de clave sin significado también resultan útiles en situaciones en las que el resto de datos están incompletos. Por ejemplo, puede asignar un número de cliente sin disponer de la dirección completa de esa persona. El resto de la aplicación puede funcionar perfectamente, y podrá agregar la información cuando la reciba. Si, como parte de la clave, la tabla ha utilizado el país o cualquier otro campo de identificación del que no disponía, correrá el riesgo de no poder utilizar la aplicación.

Así pues, por todas las razones expuestas, es recomendable utilizar un campo **Autonumérico** como campo de clave para la mayoría de las tablas. Mediante el uso de cuadros combinados y columnas ocultas, puede vincular campos al campo **Autonumérico** y ocultarlo de la vista del usuario.

6. Uso de la integridad referencial

Una vez que las tablas estén definidas y sepa el modo en que se relacionan unas con otras, asegúrese de agregar integridad referencial para reforzar la relación. Esto evita que los campos vinculados se modifiquen incorrectamente y que se dejen registros "huérfanos". El motor de bases de datos Microsoft Jet admite una sofisticada integridad referencial, que permite disponer de actualizaciones y eliminaciones en cascada. En general, el campo Id. no se debe cambiar. Por lo tanto, las actualizaciones en cascada carecen de importancia, pero las eliminaciones en cascada pueden resultar muy útiles.

Por ejemplo, si tiene una tabla de facturas relacionada con una tabla de pedidos en la que una factura puede tener un número ilimitado de pedidos (artículos de línea) y cada registro contiene el número de factura al que está vinculado, las eliminaciones en cascada permitirán eliminar el registro de factura y, automáticamente, eliminar todos los registros de pedido correspondientes. Así se asegura de no tener nunca un registro de pedido sin su correspondiente registro de factura.