Diseño de datos: Normalización

por Fernando Dodino, Juan Zaffaroni

Versión 2.0
Septiembre 2013

Antes que nada, aclaramos que vamos a trabajar la normalización y desnormalización desde el *modelo relacional*.

Normalización

Normalizar una base de datos significa transformar un conjunto de datos que tienen una cierta complejidad en su entendimiento y que su distribución en el modelo provoca problemas de lógica en las acciones de manipulación de datos, en una estructura de datos que posea un diseño claro, donde estos datos guarden coherencia y no pierdan su estado de asociación.

Objetivos de la normalización

- Reducir la redundancia de datos, y por lo tanto, las inconsistencias.
- Facilitar el mantenimiento de los datos.
- Evitar anomalías en la manipulación de datos.
- Reducir el impacto de los cambios en los datos.

A qué se refiere la normalización

- A la minimización de la redundancia de datos.
- A cómo se estructuran los atributos de cada relación.
- A que los atributos de una relación no posean estructura interna.
- A que la dependencia que se forma entre los atributos de una tupla, no se diluya en el momento de la transformación y se mantenga persistente.
- A cómo se estructuran los datos de una relación, definiendo una clave primaria para identificar a cada tupla y definiendo claves foráneas que hagan referencia a claves primarias de otras relaciones.
- A que la cantidad de atributos de una clave primaria sea mínima.
- A la creación de una estructura lógica que sea fácil de comprender y mantener.
- A facilitar el control de la manipulación de datos para que no se violen las reglas de integridad.

A qué no se refiere la normalización

- Al tipo de datos de los atributos.
- A la longitud de los atributos.
- A la nomenclatura de los atributos y de las relaciones.
- A la ubicación de los atributos dentro de la cabecera de la relación.
- A la cantidad de relaciones, o cantidad de atributos dentro de una relación.

Ej: en la siguiente relación que muestra los pedidos de pizza de un cliente

ID de pedido	**ID de cliente**	**Nombre del cliente**	**Domicilio del cliente**	**etc ...**
1	3	Blas Giunta	Almirante Brown 1048 - Isidro Casanova	
2	3	Blas Giunta	Almirante Brown 1048 - Isidro Casanova	

En subrayado aparece la clave principal de la relación.

Podemos ver que las columnas nombre y domicilio del cliente dependen funcionalmente de la columna Id de cliente. Esta definición provoca redundancia de datos: la información sobre el domicilio del cliente aparece en más de un lugar.

Formas normales

Las **formas normales** son heurísticas o criterios que permiten resolver esas redundancias. En algunas bibliografías se habla de errores o inconsistencias, es verdad que las redundancias pueden surgir por errores en el diseño de los datos, pero veremos a continuación que esa redundancia puede ser buscada.

Cada forma normal introduce restricciones nuevas, donde la primera restricción que aplica es cumplir la forma normal anterior. Entonces, cuando una tabla cumple segunda forma normal, por definición cumple primera forma normal. De la misma manera, si una tabla cumple 3FN (tercera forma normal), por definición cumple 1FN **y** 2FN.

Forma canónica

Aquí se eliminan los atributos que son el resultado de un cálculo.

En el ejemplo de las pizzas, si

- en el Pedido tenemos el total a pagar (que resultaría de la cantidad de gustos pedidos *

el valor de cada gusto),
- en el Cliente tenemos la cantidad de pedidos hechos (que se puede obtener consultando la tabla de pedidos),

estamos trabajando con atributos calculables. Otros ejemplos clásicos son guardar el total de empleados a cargo que tiene un jefe, el monto ($) que representa la deuda de un cliente, etc.

Primera forma normal

Un buen resumen puede verse curiosamente en http://es.wikipedia.org/wiki/1NF, donde debemos satisfacer estas reglas:

1. No hay orden de arriba a abajo en las filas (el pedido con ID 2 no está después del pedido con ID 1).
2. No hay orden de izquierda a derecha en las columnas (el ID de cliente no está antes que el ID del pedido).
3. No hay filas duplicadas (cada pedido aparece como una única fila en la relación Pedidos).
4. No hay campos repetitivos (no hay arrays de gustos pedidos en el Pedido).
5. Todas las columnas son regulares: esto significa que las filas se identifican únicamente a través de claves candidatas. Esto incluye claves subrogadas autoincrementales (los campos ID que veremos definidos más adelante en los ejemplos). pero *no debe haber referencias ocultas a IDs de fila, IDs de objeto (de base de datos) o datos que sirvan únicamente para hacer procesamientos temporales*[1].

La definición 4 plantea que no podemos definir grupos repetitivos. El modelo conceptual nos dice: "Un pedido se compone de uno o más pizzas, cada una de diferente gusto". En el modelo de objetos, el objeto Pedido tiene n gustos, representado por una referencia a una colección de objetos Gusto. En el modelo relacional no tenemos arrays de gustos, en el pedido no puedo hacer:

Pedido
- Id de pedido
- Id de cliente
- ...
- Gustos [1..n]

Ni puedo generar una relación que se implemente así:

[1] Para más información http://books.google.com.ar/books?id=y_eVBB5qdwMC&pg=PA107&lpg=PA107&dq=%22What+First+Normal+Form+Really+Means%22&source=bl&ots=DjN3HBuY3B&sig=P8GKhlKk4BqQywVU6NgiXjTd8wg&hl=es-419&sa=X&ei=zmmGUtC3ObPUsASnhoDAAw&ved=0CGAQ6AEwBg#v=onepage&q=%22What%20First%20Normal%20Form%20Really%20Means%22&f=false

ID de pedido	**ID de cliente**	**... etc ...**	**Gustos pedidos**
1	3		2 de Roquefort 2 de Muzzarella 1 de Fugazzetta
2	3		1 de Palmitos 1 de Fugazzetta 2 de Ananá

No se permiten atributos multivaluados.

Dependencia funcional

Dados dos atributos de una entidad, b depende funcionalmente de a si todo posible valor de a tiene asociado un único valor de b.

Se denota: a -> b.

La relación inversa no siempre se cumple.
En el siguiente ejemplo:

Id Fabricante	Id Producto	Descripción Producto	Razón Social Fabricante	...
78	65	Pirufio AA7146	EATON	
78	64	Tornoplete S40/W84	EATON	
84	65	Pirufio AA7146	TODARO	

Podemos observar que

- ID de Fabricante -> Razón Social del Fabricante.
- ID de Producto -> Descripción del producto, ID de Fabricante. Se pueden utilizar varios atributos al definir la dependencia funcional: a -> b, c.

Determinante

Es un atributo o conjunto de atributos que definen el valor de otros atributos. Siguiendo el ejemplo anterior

- ID de Fabricante
- ID de Producto

son determinantes.

Regla de normalización general

Vamos a buscar que todos los determinantes sean claves candidatas. Es decir que dada una relación R compuesta por atributos A, B, C y D donde A es clave principal, A sea determinante de B, C y D (B, C y D dependen funcionalmente de A).

R (A, B, C, D),
A -> BCD.

Si R tiene una clave compuesta, vamos a buscar que toda la clave compuesta sea determinante de los atributos no-clave.

R (A, B, C, D),
AB -> C, D.

Segunda forma normal

En el ejemplo de las pizzas, si tenemos la relación Pedido_Gusto:

ID de pedido	**ID de Gusto**	**Cantidad**	**Descripción del gusto**
1	1	2	Roquefort
1	2	2	Muzzarella
1	3	1	Fugazzetta
2	4	1	Palmitos
2	3	1	Fugazzetta
2	5	2	Ananá

Vemos que la descripción del gusto depende funcionalmente del Id de Gusto
ID de Gusto -> Descripción del gusto,
Id de Gusto es determinante de Descripción del gusto, porque el valor de descripción del gusto se puede obtener a partir de la columna Id de Gusto. Id de Gusto ¿forma parte de la clave? Sí, pero no es por sí sola clave candidata, debe ir acompañada del pedido.

Id de Pedido+Id de Gusto->Cantidad, pero
Id de Gusto -> Descripción de Gusto.

La solución es partir la relación en dos:

Pedidos_Gustos, que es una entidad-relación:

ID de pedido	**ID de Gusto**	**Cantidad**

1	1	2
1	2	2
1	3	1
2	4	1
2	3	1
2	5	2

nótese que la cantidad depende de la clave Pedido_Gusto, por lo tanto queda como columna de esta relación.

y la relación Gustos resulta una entidad independiente:

ID de Gusto	**Descripción del gusto**
1	Roquefort
2	Muzzarella
3	Fugazzetta
4	Palmitos
5	Ananá

En la tabla de Gustos quedan 5 registros, mientras que se mantienen los 6 originales en Pedido_Gusto.

Ahora sí cumplimos que todo determinante sea clave candidata:
para la relación Pedidos_Gustos: Id de Pedido+Id de Gusto->Cantidad
para la relación Gustos: Id de Gusto -> Descripción de Gusto.

Para la tribuna decimos: "*Una estructura de datos está en 2FN si y sólo si no hay dependencias funcionales entre los atributos claves (no hay dependencias parciales), y se satisface 1FN*".

Tercera forma normal

En la relación Pedido del ejemplo del delivery de pizza,

Tabla de pedidos

ID de pedido	**ID de**	**Fecha del**	**Nombre del**	**Domicilio del cliente**	**... etc**

	cliente	pedido	cliente		...
1	3	10/06/2012	Blas Giunta	Almirante Brown 1048 - Isidro Casanova	
2	3	16/09/2012	Blas Giunta	Almirante Brown 1048 - Isidro Casanova	

Id de Pedido -> Id de cliente y Fecha del pedido, lo cual está bien (Id de Pedido es determinante *y* clave principal)
Pero además
Id de Cliente -> Nombre del cliente, Domicilio del cliente
O sea, pasado a castellano, nombre y domicilio dependen del Id de Cliente, y Id de Cliente -siendo determinante- no es clave (ni principal ni alternativa).

Tenemos una redundancia, como se puede ver en la tabla misma:

- cuando Blas Giunta se muda, hay que actualizar todos los pedidos asociados a él
- también se plantea un problema al eliminarse a Blas Giunta como cliente

En definitiva, esta normalización se parece a la anterior:

- pero mientras que en el primer caso el determinante formaba parte de la clave (2FN)
- aquí el determinante es un atributo que no participa de la clave (3FN)

Ahora sí definimos 3FN: *"Una estructura de datos está en 3FN si y sólo si no hay dependencias funcionales entre los atributos no-claves (y se satisface 2FN)".* Es la extensión de la 2FN incluyendo a los atributos que no forman parte de la clave principal.

Para pasar esta solución a 3FN debemos particionar la relación original en dos tablas:

Tabla de pedidos

ID de pedido	**ID de cliente**	**Fecha del pedido**	**... etc ...**
1	3	10/06/2012	
2	3	16/09/2012	

Tabla de clientes

ID de cliente	**Nombre del cliente**	**Domicilio del cliente**	**... etc ...**
3	Blas Giunta	Almirante Brown 1048 - Isidro Casanova	

El id de cliente pasa a ser clave principal de Clientes, y permite relacionar pedido y cliente

(como una clave foránea).

Las tres primeras formas normales son las más ampliamente utilizadas en el diseño de datos, a partir de aquí comentaremos formas normales adicionales sugeridas para algunos casos.

Forma normal Boyce & Codd

Volviendo sobre el pedido, vamos a hacer algunos agregados:

- El negocio se diversificó en varias sucursales: Budge, Aldo Bonzi y Longchamps.
- Cada sucursal conserva su propia numeración (Longchamps, Budge y Aldo Bonzi tienen su propio pedido 1, pero son 3 registros diferentes, claro está)
- Queremos saber quién tomó el pedido telefónico. Cada recepcionista trabaja en una única sucursal.

Vemos cómo queda la nueva tabla de pedidos:

Sucursal	**ID de pedido**	**ID de cliente**	**Fecha del pedido**	**Atendido por**	**... etc ...**
Budge	1	3	10/06/2012	Germán	
Aldo Bonzi	1	3	16/09/2012	Nuria	

Sucursal + ID de Pedido -> Id de cliente, Fecha del pedido y Atendido por. Lo cual es razonable según la lógica de negocio.

La tabla de pedidos:

- ¿cumple 1FN? Sí, no hay grupos repetitivos, ni orden de filas o columnas.
- ¿cumple 2FN? Seguro, no hay dependencias funcionales con atributos que formen parte de la clave.
- ¿cumple 3FN? También, no hay dependencias funcionales con atributos no-clave.

Pero este caso particular nos muestra que
Atendido por -> Sucursal.
O sea, partiendo de quién nos atendió sabemos en qué sucursal se hizo el pedido.

"Atendido por" es determinante. Peeeeero... no es clave candidata por sí sola (forma clave compuesta con el ID de Pedido).

Entonces para cumplir BCNF (también llamado 3.5FN o Heath FN):

Tabla de pedidos

ID Recepcionista (ex-atendido por)	**ID de pedido**	**ID de cliente**	**Fecha del pedido**	**... etc ...**	

Germán	1	3	10/06/2012		
Nuria	1	3	16/09/2012		

Recepcionistas_Sucursales

ID de Recepcionista	**Sucursal**
Germán	Budge
Nuria	Aldo Bonzi
Malena	Aldo Bonzi

Sí, sí, las tablas de recepcionistas y de sucursales tienen identificadores raros, no usuales: en general utilizar descriptores como clave no es una buena idea, tanto por el tamaño de almacenamiento de la clave como por la posibilidad de que al tipear mal Aldo Bonsi o Aldo Bonci ese identificador quede permanentemente mal, o bien tengamos dos tuplas que estén representando el mismo dato de la realidad (Aldo Bonzi + Aldo Bonsi).

Pero como ejemplo didáctico vale:

- si se elimina el pedido 1, todavía sabemos que Germán atiende en la sucursal Budge
- no necesitamos que se registre un pedido atendido por Malena para saber que atiende en la sucursal Aldo Bonzi

¿Qué dice BCFN?

"Todo determinante debe ser clave candidata"

¡Oia! Nuestra regla común para normalizar...

Cuarta y quinta forma normal (¿no será mucho?)

Cuando tenemos dependencias multivaluadas (en relaciones muchos a muchos), aparecen redundancias que es posible eliminar. Sigamos con el ejemplo del delivery, donde tenemos la siguiente entidad:

Sucursal	**Gusto**	**Cocinero**
Budge	Espinaca	Tomás
Budge	Palmitos	Tomás
Budge	Palmitos	Lucas

Budge	Palmitos	Sofía
Aldo Bonzi	Espinaca	Ludmila
Aldo Bonzi	Espinaca	Eleuteria

¿Qué significa que hay dependencias multivaluadas?

- Que estamos diciendo 3 veces que Budge prepara el gusto Palmitos
- De la misma manera sabemos dos veces que Tomás trabaja en la sucursal Budge

Tenemos R(A, B, C) , lo separamos en R(A, B) y R2(A, C):

Sucursales_Gustos

Sucursal	**Gusto**
Budge	Espinaca
Budge	Palmitos
Aldo Bonzi	Espinaca

y

Sucursales_Cocineros

Sucursal	**Cocinero**
Budge	Tomás
Budge	Lucas
Budge	Sofía
Aldo Bonzi	Ludmila
Aldo Bonzi	Eleuteria

Si además el negocio quiere restringir que determinados gustos sólo puedan ser preparados por determinados cocineros, llegamos a la quinta forma normal: generamos una entidad que mapea el JOIN entre Cocineros-Gustos:

Gusto	**Cocinero**
Espinaca	Tomás

Palmitos	Tomás
Palmitos	Lucas
Palmitos	Sofía
Espinaca	Ludmila
Espinaca	Eleuteria

De esta manera eliminamos las dependencias multivaluadas. Se genera una R3(B, C)

¿Hay que respetar el orden para normalizar?

Por supuesto que no.

Por un tema didáctico hemos presentado las formas normales en orden. Incluso en muchas bibliografías verán que la normalización se realiza en pasos intermedios, siguiendo un algoritmo, donde se va cumpliendo primero 1FN, luego 2FN y así sucesivamente. Nuestra intención no es que sigas mecánicamente esos pasos, al fin y al cabo como ingenieros nos van a pedir que tomemos decisiones de diseño, entonces lo que queremos que tengas en cuenta a la hora de normalizar (si es que querés normalizar, ver el punto siguiente):

- ¿Hay dependencias funcionales entre los atributos de esta relación? (como verás, no importa tanto si forman parte de la clave o no, que estemos normalizando según 2FN ó 3FN es secundario, sirve sólo a los fines de categorizar una actividad)
- ¿Hay grupos repetitivos o atributos multivaluados?
- ¿Estoy siguiendo correctamente las definiciones conceptuales del negocio?
- **Pero de la misma manera que no nos interesa discutir si una determinada solución encaja en un patrón Strategy, o en un State, sino en lo que la decisión aporta a resolver el problema en ese contexto, de igual manera creemos que no tiene sentido discutir si la normalización que estamos aplicando es 2FN o BCNF, sino que es una técnica que nos ayuda a evitar estructuras de datos con información redundante**.

Desnormalización

¿Qué desventajas tenemos al normalizar?

1. Cuando un cliente se muda, perdemos el domicilio real donde llevamos el pedido. Quizás deteniéndonos a pensar el atributo no-principal dirección sí dependa funcionalmente del ID de pedido, aunque esto nos puede llevar a una larga discusión.
2. Si tenemos un gran volumen de pedidos, hacer el join con clientes para saber qué clientes hicieron pedidos en junio y julio del 2012 puede llevarnos a tener problemas de performance, aún asumiendo que hay un índice por default para el ID de cliente.

Es conveniente que al normalizar tengamos en cuenta ambas fuerzas

- la que busca simplificar la información, evitar la redundancia de datos (siguiendo los consejos Don't repeat yourself, Once and only once)
- la que busca obtener exactitud en la información, tomando en cuenta problemas de performance y volumen.

Formas de desnormalizar:

- generar una vista donde físicamente los datos estén contiguos: *materialised views*. **Problema:** es dependiente de la base de datos.
- incorporar campos redundantes para
    - almacenar la cuenta de "muchos" objetos en una relación uno-a-muchos como un atributo de la relación "uno", por ejemplo
        - en el pedido podríamos tener el total de pizzas pedidas,
        - en empleado podríamos tener la cantidad de empleados a cargo (más de cero implica que es personal jerárquico, 0 es empleado común)
    - o agregar atributos de otra relación: eso no implica quitar el campo original sino asegurarse de tener una copia histórica, por ejemplo tener el domicilio de entrega del cliente en el pedido.

a) al crear un pedido de pizza y asignarle un cliente hay que ver el dato que tiene el cliente y asignarlo en el registro de la nueva factura
b) el ejemplo es bueno porque es difícil que yo quiera modificar el cliente al que le facturo un determinado documento, pero si existiera esa posibilidad tengo que estar atento a actualizar ambos campos: cliente_id y cliente_razonSocial.
c) ¿qué pasa si el cliente cambia de razón social y tengo que volver a imprimir una factura vieja? La ley dice que debería respetar la misma información al momento de facturar, entonces hay dos opciones:

- mantener un historial de cambios de nombre del cliente (fecha desde/hasta, versión, nombre)
- o tener el nombre del cliente en la factura.

Esta puede ser una **definición del negocio (lo decide el usuario)** más que una decisión técnica, en ese caso viene bien comentarlo para que a la hora de normalizar no perdamos información.

Para practicar

Tienen esta práctica para jugar en sus casas.