LA NORMALIZACION

La normalización no es en realidad una parte del diseño, sino más bien una herramienta de verificación. Si hemos diseñado bien los módelos conceptual y lógico de nuestra bases de datos, veremos que la normalización generalmente no requerirá cambios en nuestro diseño.

Normalización

Antes de poder aplicar el proceso de normalización, debemos asegurarnos de que estamos trabajando con una base de datos relacional, es decir, que cumple con la definición de base de datos relacional.

El proceso de normalización consiste en verificar el cumplimiento de ciertas reglas que aseguran la eliminación de redundancias e inconsistencas. Esto se hace mediante la aplicación de ciertos procedimientos y en ocasiones se traduce en la separación de los datos en diferentes relaciones. Las relaciones resultantes deben cumplir ciertas características:

- Se debe conservar la información.
- Conservación de los atributos.
- Conservación de las tuplas, evitando la aparición de tuplas que no estaban en las relaciones originales.
- Se deben conservar las dependencias.

Este proceso se lleva a cabo aplicando una serie de reglas llamadas *"formas normales"*.

Estas reglas permiten crear bases de datos libres de redundancias e inconsistencias, que se ajusten a la definición del doctor Codd de base de datos relacional.

Primera forma normal (1FN)

Definición: Para que una base de datos sea 1FN, es decir, que cumpla la primera forma normal, cada columna debe ser atómica.

Atómica significa "indivisible", es decir, cada *atributo* debe contener un único valor del *dominio*. Los atributos, en cada tabla de una base de datos 1FN, no pueden tener listas o arrays de valores, ya sean del mismo dominio o de dominios diferentes. Además, cada atributo debe tener un nombre único.

Tampoco pueden existir tuplas idénticas. Esto puede parecer obvio, pero no siempre es así. Supongamos una base de datos para la gestión de la biblioteca, y que el mismo día, y al mismo socio, se le presta dos veces el mismo libro (evidentemente, el libro es devuelto entre cada préstamo, claro). Esto producirá, si no tenemos cuidado, dos registros iguales. Debemos evitar este tipo de situaciones, por ejemplo, añadiendo una atributo con un identificador único de préstamo o incorporar la fecha como parte de la clave.

Como vemos, las restricciones de la primera forma normal coinciden con las condiciones de las relaciones de una base de datos relacional, por lo tanto, siempre es obligatorio aplicar esta forma normal.

Aplicar la primera forma normal es muy simple, bastará con dividir cada columna no atómica en tantas columnas atómicas como sea necesario. Por ejemplo, si tenemos una relación que contiene la información de una agenda de amigos con este *esquema*:

Agenda(Nombre, email)

El nombre, normalmente, estará compuesto por el tratamiento (señor, señora, don, doña, excelencia, alteza, señoría, etc), un nombre de pila y los apellidos. Podríamos considerar el nombre como un dato atómico, pero puede interesarnos separar algunas de las partes que lo componen.

¿Y qué pasa con la dirección de correo electrónico? También podemos considerar que es un valor no atómico, la parte a la izquierda del símbolo @ es el usuario, y a la derecha el dominio. De nuevo, dependiendo de las necesidades del cliente o del uso de los datos, podemos estar interesados en dividir este campo en dos, o incluso en tres partes (puede interesar separar la parte a la derecha del punto en el dominio).

Tanto en esta forma normal, como en las próximas que veremos, es importante no llevar el proceso de normalización demasiado lejos. Se trata de facilitar el trabajo y evitar problemas de redundancia e integridad, y no de lo contrario. Debemos considerar las ventajas o necesidades de aplicar cada norma en cada caso, y no excedernos por intentar aplicar las normas demasiado al pie de la letra.

El esquema de la relación puede quedar como sigue:

Agenda(Nombre_Tratamiento, Nombre_Pila, Nombre_Apellidos, email)

Otro caso frecuente de relaciones que no cumplen 1FN es cuando existen atributos multivaluados, si todos los valores se agrupan en un único atributo:

Libros(Titulo, autores, fecha, editorial)

Hemos previsto, muy astutamente, que un libro puede tener varios autores. No es que sea muy frecuente pero sucede, y más con libros técnicos y libros de texto.

Sin embargo, esta relación no es 1FN, ya que en la columna de autores sólo debe existir un valor del dominio, por lo tanto debemos convertir ese atributo en uno multivaluado:

Libros

Titulo	autor	fecha	editorial
Que bueno es MySQL	fulano	12/10/2003	La buena
Que bueno es MySQL	mengano	12/10/2003	La buena
Catástrofes naturales	tulano	18/03/1998	Penútriga

Dependencias funcionales

Ya hemos comentado que una relación se compone de atributos y dependencias. Los atributos son fáciles de identificar, ya que forman parte de la estructura de la relación, y además, los elegimos nosotros mismos como diseñadores de la base de datos.

Pero no es tan sencillo localizar las dependencias, ya que requieren un análisis de los atributos, o con más precisión, de las interrelaciones entre atributos, y frecuentemente la intuición no es suficiente a la hora de encontrar y clasificar todas las dependencias.

La teoría nos puede ayudar un poco en ese sentido, clasificando las dependencias en distintos tipos, indicando qué características tiene cada tipo.

Para empezar, debemos tener claro que las dependencias se pueden dar entre atributos o entre subconjuntos de atributos.

Estas dependencias son consecuencia de la estructura de la base de datos y de los objetos del mundo real que describen, y no de los valores actualmente almacenados en cada relación. Por ejemplo, si tenemos una relación de vehículos en la que almacenamos, entre otros atributos, la cilindrada y el color, y en un determinado momento todos los vehículos con 2000 c.c. son de color rojo, no podremos afirmar que existen una dependencia entre el color y la cilindrada. Debemos suponer que esto es sólo algo casual.

Para buscar dependencias, pues, no se deben analizar los datos, sino los entes a los que se refieren esos datos.

Definición: Sean X e Y subconjuntos de atributos de una relación. Diremos que Y tiene una dependencia funcional de X, o que X determina a Y, si cada valor de X tiene asociado siempre un único valor de Y.

El hecho de que X determine a Y no quiere decir que conociendo X se pueda conocer Y, sino que en la relación indicada, cada vez que el atributo X tome un determinado valor, el atributo Y en la misma tupla siempre tendrá el mismo valor.

Por ejemplo, si tenemos una relación con clientes de un hotel, y dos de sus atributos son el número de cliente y su nombre, podemos afirmar que el nombre tiene una dependencia funcional del número de

cliente. Es decir, cada vez que en una tupla aparezca determinado valor de número de cliente, es seguro que el nombre de cliente será siempre el mismo.

La dependencia funcional se representa como X → Y.

El símbolo → se lee como "implica" o "determina", y la dependencia anterior se lee como *X implica Y* o *X determina Y*.

Podemos añadir otro símbolo a nuestra álgebra de dependencias: el símbolo | significa negación. Así X → |Y se lee como *X no determina Y*.

Dependencia funcional completa

Definición: En una dependencia funcional X → Y, cuando X es un conjunto de atributos, decimos que la dependencia funcional es completa , si sólo depende de X, y no de ningún subconjunto de X.

La dependencia funcional se representa como X => Y.

Dependencia funcional elemental

Definición: Si tenemos una dependencia completa X => Y, diremos que es una dependencia funcional elemental si Y es un atributo, y no un conjunto de ellos.

Estas son las dependencias que buscaremos en nuestras relaciones. Las dependencias funcionales elementales son un caso particular de las dependencias completas.

Dependencia funcional trivial

Definición: Una dependencia funcional A → B es trivial cuando B es parte de A. Esto sucede cuando A es un conjunto de atributos, y B es a su vez un subconjunto de A.

Segunda forma normal (2FN)

Definición: Para que una base de datos sea 2FN primero debe ser 1FN, y además todas las columnas que formen parte de una *clave candidata* deben aportar información sobre la clave completa.

Esta regla significa que en una relación sólo se debe almacenar información sobre un tipo de entidad, y se traduce en que los atributos que no aporten información directa sobre la clave principal deben almacenarse en una relación separada.

Lo primero que necesitamos para aplicar esta forma normal es identificar las claves candidatas.

Además, podemos elegir una clave principal, que abreviaremos como **PK**, las iniciales de Primary Key. Pero esto es optativo, el modelo relacional no obliga a elegir una clave principal para cada relación, sino tan sólo a la existencia de al menos una clave candidata.

La inexistencia de claves candidatas implica que la relación no cumple todas las normas para ser parte de una base de datos relacional, ya que la no existencia de claves implica la repetición de tuplas.

En general, si no existe un candidato claro para la clave principal, crearemos una columna específica con ese propósito.

Veamos cómo aplicar esta regla usando un ejemplo. En este caso trataremos de guardar datos relacionados con la administración de un hotel.

Planteemos, por ejemplo, este esquema de relación:

Ocupación(No_cliente, Nombre_cliente, No_habitación, precio_noche, tipo_habitación, fecha_entrada)

Lo primero que tenemos que hacer es buscar las posibles claves candidatas. En este caso sólo existe una posibilidad:

(No_habitación, fecha_entrada)

Recordemos que cualquier clave candidata debe identificar de forma unívoca una clave completa. En este caso, la clave completa es la ocupación de una habitación, que se define por dos parámetros: la habitación y la fecha de la ocupación.

Es decir, dos ocupaciones son diferentes si cambian cualquiera de estos parámetros. La misma persona puede ocupar varias habitaciones al mismo tiempo o la misma habitación durante varios días o en diferentes periodos de tiempo. Lo que no es posible es que varias personas ocupen la misma habitación al mismo tiempo (salvo que se trate de un acompañante, pero en ese caso, sólo una de las personas es la titular de la ocupación).

El siguiente paso consiste en buscar columnas que no aporten información directa sobre la clave completa: la ocupación. En este caso el precio por noche de la habitación y el tipo de habitación (es decir, si es doble o sencilla), no aportan información sobre la clave principal. En realidad, estos datos no son atributos de la ocupación de la habitación, sino de la propia habitación.

El número de cliente y su nombre aportan datos sobre la ocupación, aunque no formen parte de la clave completa.

Expresado en forma de dependencias se ve muy claro:

(No_habitación, fecha_entrada) -> No_cliente
(No_habitación, fecha_entrada) -> Nombre_cliente
No_habitación -> precio_noche
No_habitación -> tipo_habitación

El último paso consiste en extraer los atributos que no forman parte de la clave a otra relación. En nuestro ejemplo tendremos dos relaciones: una para las ocupaciones y otra para las habitaciones:

Ocupación(No_cliente, Nombre_cliente, No_habitación, fecha_entrada**(PK)**)
Habitación(No_habitación, precio_noche, tipo_habitación)

La segunda tabla tiene una única clave candidata, que es el número de habitación. El resto de los atributos no forman parte de la clave completa (la habitación), pero aportan información sólo y exclusivamente sobre ella.

Estas dos relaciones están interrelacionadas por la clave de habitación. Para facilitar el establecimiento de esta interrelación elegiremos la clave candidata de la habitación en clave principal. El mismo atributo en la relación de ocupaciones es, por lo tanto, una clave foránea:

Ocupación(No_cliente, Nombre_cliente, No_habitación, fecha_entrada**(PK)**)
Habitación(No_habitación**(PK)**, precio_noche, tipo_habitación)

Como norma general debemos volver a aplicar la primera y segunda forma normal a estas nuevas tablas. La primera sólo en el caso de que hallamos añadido nuevas columnas, la segunda siempre.

La interrelación que hemos establecido es del tipo uno a muchos, podemos elegir una clave de habitación muchas veces, tantas como veces se ocupe esa habitación.

Dependencia funcional transitiva

Definición: Supongamos que tenemos una relación con tres conjuntos de atributos: X, Y y Z, y las siguientes dependencias X → Y, Y → Z, Y → |X. Es decir X determina Y e Y determina Z, pero Y no determina X. En ese caso, decimos que Z tiene dependencia transitiva con respecto a X, a través de Y.

Intentaremos aclarar este concepto tan teórico con un ejemplo. Si tenemos esta relación:

Ciudades(ciudad, población, superficie, renta, país, continente)

Los atributos como *población*, *superficie* o *renta* tienen dependencia funcional de *ciudad*, así que de momento no nos preocupan.

En esta relación podemos encontrar también las siguientes dependencias:

ciudad -> país, país → continente. Además, país → |ciudad. Es decir, cada *ciudad* pertenece a un *país* y cada *país* a un *continente*, pero en cada *país* puede haber muchas *ciudades*. En este caso *continente* tiene una dependencia funcional transitiva con respecto a *ciudad*, a través de *país*. Es decir, cada *ciudad* está en un *país*, pero también en un *continente*. (¡Ojo! las dependencias transitivas no son siempre tan evidentes).

Tercera forma normal 3FN

La tercera forma normal consiste en eliminar las dependencias transitivas.

Definición: Una base de datos está en 3FN si está en 2FN y además todas las columnas que no sean claves dependen de la clave completa de forma no transitiva.

Pero esto es una definición demasiado teórica. En la práctica significa que se debe eliminar cualquier relación que permita llegar a un mismo dato de dos o más formas diferentes.

Tomemos el ejemplo que usamos para ilustrar las dependencias funcionales transitivas. Tenemos una tabla donde se almacenen datos relativos a ciudades, y una de las columnas sea el país y otra el continente al que pertenecen. Por ejemplo:

Ciudades(ID_ciudad**(PK)**, Nombre, población, superficie, renta, país, continente)

Un conjunto de datos podría ser el siguiente:

Ciudades

ID_ciudad	Nombre	población	superficie	renta	país	continente
1	Paris	6000000	15	1800	Francia	Europa
2	Lion	3500000	9	1600	Francia	Europa
3	Berlin	7500000	16	1900	Alemania	Europa
4	Pekin	19000000	36	550	China	Asia
5	Bonn	6000000	12	1900	Alemania	Europa

Podemos ver que para cada aparición de un determinado país, el continente siempre es el mismo. Es decir, existe una redundancia de datos, y por lo tanto, un peligro de integridad.

Existe una relación entre país y continente, y ninguna de ellas es clave candidata. Por lo tanto, si queremos que esta table sea 3FN debemos separar esa columna:

Ciudades(ID_ciudad**(PK)**, Nombre, población, superficie, renta, nombre_pais)
Paises(nombre_pais**(PK)**, nombre_continente)

Ciudades

ID_ciudad	Nombre	población	superficie	renta	país
1	Paris	6000000	15	1800	Francia
2	Lion	3500000	9	1600	Francia
3	Berlin	7500000	16	1900	Alemania
4	Pekin	19000000	36	550	China
5	Bonn	6000000	12	1900	Alemania

Paises

país	continente
Francia	Europa
Alemania	Europa
China	Asia

Esta separación tendría más sentido si la tabla de paises contuviese más información, tal como está no tiene mucho sentido separar estas tablas, aunque efectivamente, se evita redundancia.

Forma Normal de Boyce/Codd (FNBC)

Definición: Una relación está en FNBC si cualquier atributo sólo facilita información sobre claves candidatas, y no sobre atributos que no formen parte de ninguna clave candidata.

Esto significa que no deben existir interrelaciones entre atributos fuera de las claves candidatas.

Para ilustrar esta forma normal volvamos a uno de nuestros ejemplos anteriores, el de las ocupaciones de habitaciones de un hotel.

Ocupación(No_cliente, Nombre_cliente, No_habitación, fecha_entrada)
Habitación(No_habitación**(PK)**, precio_noche, tipo_habitación)

En la primera relación los atributos No_cliente y Nombre_cliente sólo proporcionan información entre ellos mutuamente, pero ninguno de ellos es una clave candidata.

Intuitivamente ya habremos visto que esta estructura puede producir redundancia, sobre todo en el caso de clientes habituales, donde se repetirá la misma información cada vez que el mismo cliente se aloje en el hotel.

La solución, como siempre, es simple, y consiste en separar esta relación en dos diferentes:

Ocupación(No_cliente, No_habitación, fecha_entrada)
Cliente(No_cliente**(PK)**, Nombre_cliente)
Habitación(No_habitación**(PK)**, precio_noche, tipo_habitación)

Atributos multivaluados

Definición: se dice que un atributo es multivaluado cuando para una misma entidad puede tomar varios valores diferentes, con independencia de los valores que puedan tomar el resto de los atributos.

Se representa como X ->-> Y, y se lee como *X multidetermina Y*.

Un ejemplo claro es una relación donde almacenemos contactos y números de teléfono:

Agenda(nombre, fecha_nacimiento, estado_civil, teléfono)

Para cada *nombre* de la agenda tendremos, en general, varios números de *teléfono*, es decir, que *nombre* multidetermina *teléfono*: nombre ->-> teléfono.

Además, *nombre* determina funcionalmente otros atributos, como la *fecha_nacimiento* o *estado_civil*.

Por otra parte, la clave candidata no es el nombre, ya que debe ser unívoca, por lo tanto debe ser una combinación de nombre y teléfono.

En esta relación tenemos las siguientes dependencias:

- nombre -> fecha_nacimiento
- nombre -> estado_civil
- nombre ->-> teléfono, o lo que es lo mismo (nombre,teléfono) -> teléfono

Es decir, la dependencia multivaluada se convierte, de hecho, en una dependencia funcional trivial.

Este tipo de atributos implica redundancia ya que el resto de los atributos se repiten tantas veces como valores diferentes tenga el atributo multivaluado:

Agenda

nombre	fecha_nacimiento	estado_civil	teléfono
Mengano	15/12/1985	soltero	12322132
Fulano	13/02/1960	casado	13321232
Fulano	13/02/1960	casado	25565445

Fulano	13/02/1960	casado	36635363
Tulana	24/06/1975	soltera	45665456

Siempre podemos evitar el problema de los atributos multivaluados separandolos en relaciones distintas. En el ejemplo anterior, podemos crear una segunda relación que contenga el nombre y el teléfono:

Agenda(nombre**(PK)**, fecha_nacimiento, estado_civil)
Teléfonos(nombre, teléfono**(PK)**)

Para los datos anteriores, las tablas quedarían así:

Agenda

nombre	fecha_nacimiento	estado_civil
Mengano	15/12/1985	soltero
Fulano	13/02/1960	casado
Tulana	24/06/1975	soltera

Teléfonos

nombre	teléfono
Mengano	12322132
Fulano	13321232
Fulano	25565445
Fulano	36635363
Tulana	45665456

Ejemplos de Normalización

Ejemplo 1

Aplicaremos ahora la normalización a las relaciones que obtuvimos en el capítulo anterior.

En el caso del primer ejemplo, para almacenar información sobre estaciones meteorológicas y las muestras tomadas por ellas, habíamos llegado a esta estructura:

Estación(Identificador, Latitud, Longitud, Altitud)
Muestra(IdentificadorEstacion, Fecha, Temperatura mínima, Temperatura máxima, Precipitaciones, Humedad mínima, Humedad máxima, Velocidad del viento mínima, Velocidad del viento máxima)

Primera forma normal

Esta forma nos dice que todos los atributos deben ser atómicos.

Ya comentamos antes que este criterio es en cierto modo relativo, lo que desde un punto de vista puede ser atómico, puede no serlo desde otro.

En lo que respecta a la relación *Estación*, el Identificador y la Altitud son claramente atómicos. Sin embargo, la Latitud y Longitud pueden considerarse desde dos puntos de vista. En uno son coordenadas (de hecho, podríamos haber considerado la posición como atómica, y fundir ambos atributos en uno). A pesar de que ambos valores se expresen en grados, minutos y segundos, más una orientación, norte, sur, este u oeste, puede hacernos pensar que podemos dividir ambos atributos en partes más simples.

Esta es, desde luego, una opción. Pero todo depende del uso que le vayamos a dar a estos datos. Para nuestra aplicación podemos considerar como atómicos estos dos atributos tal como los hemos definido.

Para la relación *Muestras* todos los atributos seleccionados son atómicos.

Segunda forma normal

Para que una base de datos sea 2FN primero debe ser 1FN, y además todas las columnas que formen parte de una clave candidata deben aportar información sobre la clave completa.

Para la relación *Estación* existen dos claves candidatas: identificador y la formada por la combinación de Latitud y Longitud.

Hay que tener en cuenta que hemos creado el atributo Identificador sólo para ser usado como clave principal. Las dependencias son:

Identificador -> (Latitud, Longitud)
Identificador -> Altitud

En estas dependencias, las claves candidatas Identificador y (Latitud, Longitud) son equivalentes y, por lo tanto, intercambiables.

Esta relación se ajusta a la segunda forma normal, veamos la de *Muestras*.

En esta relación la clave principal es la combinación de (Identificador, Fecha), y el resto de los atributos son dependencias funcionales de esta clave. Por lo tanto, también esta relación está en 2FN.

Tercera forma normal

Una base de datos está en 3FN si está en 2FN y además todas las columnas que no sean claves dependen de la clave completa de forma no transitiva.

No existen dependencias transitivas, de modo que podemos afirmar que nuestra base de datos está en 3FN.

Forma normal de Boyce/Codd

Una relación está en FNBC si cualquier atributo sólo facilita información sobre claves candidatas, y no sobre atributos que no formen parte de ninguna clave candidata.

Tampoco existen atributos que den información sobre otros atributos que no sean o formen parte de claves candidatas.

Cuarta forma normal

Esta forma se refiere a atributos multivaluados, que no existen en nuestras relaciones, por lo tanto, también podemos considerar que nuestra base de datos está en 4FN.

Ejemplo 2

Nuestro segundo ejemplo se modela una biblioteca, y su esquema de relaciones final es este:

Libro(ClaveLibro, Título, Idioma, Formato, ClaveEditorial)
Tema(ClaveTema, Nombre)
Autor(ClaveAutor, Nombre)
Editorial(ClaveEditorial, Nombre, Dirección, Teléfono)
Ejemplar(ClaveEjemplar, ClaveLibro, NúmeroOrden, Edición, Ubicación, Categoría)
Socio(ClaveSocio, Nombre, Dirección, Teléfono, Categoría)
Préstamo(ClaveSocio, ClaveEjemplar, NúmeroOrden, Fecha_préstamo, Fecha_devolución, Notas)
Trata_sobre(ClaveLibro, ClaveTema)
Escrito_por(ClaveLibro, ClaveAutor)

Los ejemplos que estamos siguiendo desde el capítulo 2 demuestran que un buen diseño conceptual sirve para diseñar bases de datos relacionales libres de redundancias, y generalmente, la normalización no afecta a su estructura.

Este ha sido el caso del primer ejemplo, y como veremos, también del segundo.

Primera forma normal

Tenemos, desde luego, algunos atributos que podemos considerar no atómicos, como el nombre del autor, la dirección de la editorial, el nombre del socio y su dirección. Como siempre, dividir estos atributos en otros es una decisión de diseño, que dependerá del uso que se le vaya a dar a estos datos. En nuestro caso, seguramente sea suficiente dejarlos tal como están, pero dividir algunos de ellos no afecta demasiado a las relaciones.

Segunda forma normal

Para que una base de datos sea 2FN primero debe ser 1FN, y además todas las columnas que formen parte de una clave candidata deben aportar información sobre la clave completa.

En el caso de *Libro*, la única clave candidata es ClaveLibro. Todos los demás valores son repetibles, pueden existir libros con el mismo título y de la misma editorial editados en el mismo formato e idioma. Es decir, no existe ningún otro atributo o conjunto de atributos que puedan identificar un libro de forma unívoca.

Se pueden dar casos especiales, como el del mismo libro escrito en diferentes idiomas. En ese caso la clave será diferente, de modo que los consideraremos como libros distintos. Lo mismo pasa si el mismo libro aparece en varios formatos, o ha sido editado por distintas editoriales.

Es decir, todos los atributos son dependencias funcionales de ClaveLibro.

Con *Tema* y *Autor* no hay dudas, sólo tienen dos atributos, y uno de ellos ha sido creado específicamente para ser usado como clave.
Los tres atributos de *Editorial* también tienen dependencia funcional de ClaveEditorial.
Y lo mismo cabe decir para las entidades *Ejemplar*, *Socio* y *Préstamo*.

En cuanto a las relaciones que almacenan interrelaciones, la clave es el conjunto de todos los atributos, de modo que todas las dependencias son funcionales y triviales.

Tercera forma normal

Una base de datos está en 3FN si está en 2FN y además todas las columnas que no sean claves dependen de la clave completa de forma no transitiva.

En *Libro* no hay ningún atributo que tenga dependencia funcional de otro atributo que no sea la clave principal. Todos los atributos defienen a la entidad *Libro* y a ninguna otra.

Las entidades con sólo dos atributos no pueden tener dependencias transitivas, como *Tema* o *Autor*.

Con *Editorial* tampoco existen, todos los atributos dependen exclusivamente de la clave principal.
En el caso del *Ejemplar* tampoco hay una correspondencia entre ubicación y edición. O al menos no podemos afirmar que exista una norma universal para esta correspondencia. Es posible que todas las primeras ediciones se guarden en el mismo sitio, pero esto no puede ser una condición de diseño para la base de datos. Del mismo modo que podríamos almacenar todos los ejemplares de la misma categoría en el mismo sitio, y esto tampoco debe ser una condición de diseño.

Y para *Préstamo* los tres atributos que no forman parte de la clave candidata se refieren sólo a la entidad *Préstamo*.

Forma normal de Boyce/Codd

Una relación está en FNBC si cualquier atributo sólo facilita información sobre claves candidatas, y no sobre atributos que no formen parte de ninguna clave candidata.

Tampoco existen atributos que den información sobre otros atributos que no sean o formen parte de claves candidatas.