Tema 5. Diseño lógico de bases de datos relacionales

A la hora de diseñar una base de datos relacional, podemos usar dos enfoques distintos. Por una parte, y tal y como hemos visto hasta ahora, podemos usar un modelo conceptual, como el Entidad-Relación, lo que ofrece una representación lógica de la realidad, y luego aplicarle una transformación para obtener el conjunto de tablas que forman el esquema lógico de la base de datos relacional. Por otra parte, podemos usar un método específico para el diseño de bases de datos relacionales, como es la normalización. La normalización, a diferencia de la transformación en tablas de un diagrama E-R, permite dar una medida formal de por qué el diseño de una tabla es mejor que otro. En este tema estudiaremos el proceso de normalización como método específico para el diseño de bases de datos relacionales.

5.1 Consejos de diseño para esquemas de relación

El diseño de bases de datos puede enfocarse con una metodología ascendente o descendente. Con una metodología *ascendente* se consideran primero las relaciones entre atributos individuales, y luego se utilizan estas relaciones básicas para crear las tablas. Este enfoque no se usa mucho en la práctica, ya que tiene el inconveniente de presentar un gran número de relaciones de atributos como punto de partida. En cambio, una metodología *descendente* parte de varias agrupaciones de atributos en tablas que ya se han obtenido en el diseño conceptual y la transformación al modelo de datos. Posteriormente, se aplica *diseño por análisis* a las tablas de forma individual y conjunta, obteniendo una mayor descomposición hasta que se satisfacen todas las propiedades deseadas.

En esta sección vamos a analizar cuatro *medidas informales* de calidad para el diseño de tablas o esquemas de relación: la semántica de los atributos, la reducción de valores redundantes en las filas, la reducción de valores nulos en las filas y la supresión de la posibilidad de generación de filas incorrectas.

5.1.1. Semántica de la relación

Cuando agrupamos atributos para formar un esquema de relación, suponemos que hay un *significado* asociado a los atributos. Este significado o *semántica* especifica como se interpretan los valores de los atributos de una tupla de la relación; es decir, qué relación existe entre los valores de los atributos de una tupla. Con un diseño conceptual

adecuado y una transformación en relaciones que no pierda esta semántica, el diseño resultante debería estar claro.

Por ejemplo, consideremos los siguientes esquemas de relación:

EMPLEADO (NombreEmp, NumSS, FechaEntrada, Direccion, NumDep)

DEPARTAMENTO (NumDep, NombreDep, NumSSJefe)

Está claro que cada tupla de EMPLEADO representa a un empleado, con información sobre su nombre, número de la SS, fecha de entrada en la empresa, dirección y número del departamento al que pertenece (NumDep), que es una clave externa que representa una *relación implícita* entre EMPLEADO y DEPARTAMENTO.

Basándonos en el significado de las relaciones, es recomendable diseñar un esquema de relación de modo que su significado resulte fácil de explicar. No debemos combinar atributos de varios conjuntos de entidades o de relaciones en una única relación. Si lo hacemos, la relación tiende a ser una mezcla de varias entidades y relaciones, y será semánticamente confusa.

Consideremos el siguiente esquema:

EMPLEADO_DPTO (NombreEmp, NumSS, FechaEntrada, Direccion, NumDep, NombreDep, NumSSJefe)

La semántica de este esquema también está clara, pero no cumple la recomendación hecha, ya que mezcla atributos de empleados y departamentos. Resulta útil como vista, pero puede dar problemas si se usa como relación de base.

5.1.2. Redundancia y Anomalías en la actualización

Uno de los objetivos a tener en cuenta en el diseño del esquema de una base de datos es reducir el espacio de almacenamiento ocupado por los datos. La forma de agrupación de los atributos en los esquemas de relación tiene un efecto importante sobre el espacio de almacenamiento. Por ejemplo, supongamos que tenemos una base de datos con información sobre empleados y departamentos, en la que la información del departamento de un empleado está en la misma tabla que la información sobre el empleado:

EMPLEADO_DPTO (NombreEmp, NumSS, FechaEntrada, Direccion, NumDep, NombreDep, NumSSJefe)

En este ejemplo podemos ver que la información sobre un departamento se repite para todos los empleados que pertenecen a ese departamento, tal y como se muestra a continuación.

NombreEmp	NumSS	FechaEntrada	Direccion	NumDep	NombreDep	NumSSJefe
Javier Ros	1	10-10-1990	Azul 6	010	Ventas	5
Isabel Gil	2	16-12-1991	Gris 5	010	Ventas	5

Inma Sol	5	1-2-1990	Jaspe 3	010	Ventas	-
Jose Leal	7	6-7-1998	Pais 45	011	Compras	7

Podemos plantearnos un esquema alternativo:

EMPLEADO (NombreEmp, NumSS, FechaEntrada, Direccion, NumDep)

DEPARTAMENTO (NumDep, NombreDep, NumSSJefe)

En este esquema, los datos del departamento se almacenan una única vez, con lo que se reduce el espacio necesario para el almacenamiento.

EMPLEADOS

NombreEmp	NumSS	FechaEntrada	Direccion	NumDep	NumSSJefe
Javier Ros	1	10-10-1990	Azul 6	010	5
Isabel Gil	2	16-12-1991	Gris 5	010	5
Inma Sol	5	1-2-1990	Jaspe 3	010	-
Jose Leal	7	6-7-1998	Pais 45	011	

DPTO

NumDep	NombreDep
010	Ventas
011	Compras

La necesidad de más espacio de almacenamiento no es el único problema derivado de un diseño incorrecto. La redundancia introducida (repetición de información) trae consigo otro tipo de problemas que pueden aparecer al actualizar los datos, y que se conocen como *anomalías de actualización*.

Existen tres tipos de anomalías de actualización, conocidas como anomalías de inserción, anomalías de eliminación, y anomalías de modificación.

Las anomalías de inserción pueden ser a su vez de dos clases, que vamos a mostrar sobre el ejemplo anterior

En primer lugar, al insertar un empleado en la tabla EMPLEADO_DPTO, necesitamos incluir los valores de los atributos del departamento al que pertenece, o valores nulos si aún no está asignado a ningún departamento. Además, al incluir los valores de atributos de un departamento, estos tienen que concordar con los valores correspondientes a ese departamento en otras filas de la tabla. En el esquema que divide esta tabla, no tenemos este problema de concordancia, porque los datos están almacenados una sola vez por departamento en la tabla DEPARTAMENTO.

En segundo lugar, tenemos problemas para insertar en la tabla `EMPLEADO_DPTO` un nuevo departamento que no tenga asignados aún empleados. Sólo podemos hacerlo introduciendo valores nulos en los atributos correspondientes al empleado, y resulta que la clave primaria de la tabla es el número de seguridad social del empleado. Además, al insertar el primer empleado habría que borrar esta fila inicial. En el esquema dividido no tenemos este problema, porque las filas se almacenan en tablas distintas.

Por lo que respecta a las anomalías de eliminación, si eliminamos de `EMPLEADO_DPTO` la fila del último empleado de un departamento, perderemos la información de ese departamento. Esto lo podemos ver ilustrado en el ejemplo anterior, en el que si eliminamos el registro del empleado `José Leal`, también perderemos la información del departamento de `Compras`. Este problema no existe en el esquema dividido.

Las anomalías de modificación pueden ilustrarse con este ejemplo. Si tenemos que modificar un atributo de un departamento en `EMPLEADO_DPTO` (por ejemplo, el número de seguridad social de su jefe, NumSSJefe), tendremos que actualizar las filas de todos los empleados de ese departamento, porque si no, la base de datos quedaría en un estado inconsistente, es decir, habría dos jefes distintos para un departamento, y solo puede haber uno. Esto no sucede con el esquema dividido.

En función de las anomalías estudiadas, es recomendable diseñar los esquemas de las relaciones de modo que no presenten anomalías de inserción, eliminación o modificación. En caso de que las haya, deben marcarse claramente para que los programas que actualicen la base de datos funcionen correctamente.

Esta recomendación puede verse como un refinamiento de la hecha en el punto anterior, y para seguirla puede ser conveniente recurrir a un mecanismo formal para su comprobación. También hay casos en los que deben violarse estas recomendaciones para obtener un mejor rendimiento en ciertas consultas, aunque esto puede resolverse con vistas.

5.1.3. Valores nulos en las tuplas

Hay ocasiones en las que agrupamos muchos atributos en una relación, y luego no son aplicables a todas las tuplas. Esto puede acarrear un almacenamiento supérfluo y complicar la comprensión de los atributos y la especificación de las operaciones de combinación de relaciones. Los valores nulos también resultan problemáticos al usar funciones de agregación.

Los valores nulos pueden tener varias interpretaciones:

- El atributo no es aplicable a esa fila
- El valor del atributo para esa fila es desconocido
- El valor se conoce pero no se ha registrado todavía

Representar todos los nulos del mismo modo puede hacer que se confundan sus posibles significados, por lo que es recomendable minimizar la presencia de valores

nulos en una base de datos relacional, y si es necesario emplearlos, hay que comprobar que sólo se aplican excepcionalmente, y no a la mayoría de tuplas de una relación.

5.1.4. Generación de tuplas incorrectas

Generalmente, la conexión entre distintos esquemas de relación se produce a través de claves primarias y claves externas, pero puede haber casos en los que relacionemos esquemas de relación con atributos que no son claves primarias ni externas. En este caso, podemos tener problemas al combinar las relaciones mediante producto natural, ya que pueden generarse tuplas incorrectas supérfluas, y no obtendremos la información original.

Por lo tanto, es recomendable diseñar los esquemas de relación de forma que puedan combinarse mediante condiciones de igualdad sobre atributos que sean claves primarias o externas, para garantizar que no se forman tuplas incorrectas. No se deben incluir relaciones con atributos coincidentes que no sean combinaciones de una clave externa con una clave primaria. En el caso de que sea necesario incluir estas relaciones, no deben combinarse usando estos atributos, ya que la combinación puede generar tuplas incorrectas.

5.2. Dependencias funcionales

El concepto de dependencia funcional es el más importante para diseñar esquemas de bases de datos relacionales, ya que es la base de la definición de propiedades que deben cumplir estos esquemas en el proceso de diseño mediante normalización.

5.2.1. Definición de dependencia funcional

Una dependencia funcional indica una restricción entre dos conjuntos de atributos de una base de datos. Supongamos que tenemos un esquema de una base de datos relacional con n atributos $A_1, A_2 \dots A_n$, y que toda la base de datos está dentro de una *relación universal* $R = \{ A_1, A_2 \dots A_n\}$.

Una dependencia funcional, representada como $X \rightarrow Y$, entre dos conjuntos de atributos X e Y que son subconjuntos de R especifica una *restricción* entre los valores de las posibles tuplas de un estado o instancia r de R. La restricción dice que para cualquier par de tuplas t_1 y t_2 de r, tales que $t_1[X] = t_2[X]$, se cumple que $t_1[Y] = t_2[Y]$. Esto implica que los valores del componente Y de una tupla dependen de los valores del componente X, y que los valores del componente X determinan de manera única los valores del componente Y. También se dice que Y depende funcionalmente de X. El conjunto de atributos X se denomina *parte izquierda* de la DF, y el conjunto Y es la *parte derecha*. Debemos hacer dos observaciones:

- Si R tiene una restricción que indica que no puede haber más de una tupla con un valor X, es decir, X es una *clave candidata* de R, entonces $X \rightarrow Y$ para cualquier subconjunto de atributos Y de R.
- Si $X \rightarrow Y$ en R, esto no da información sobre si $Y \rightarrow X$ en R o no.

Las dependencias funcionales son propiedades del *significado* de los atributos, y no se pueden extraer directamente a partir de su contenido. Es decir, puede que en un momento dado no haya dos empleados con el mismo nombre, pero nada impide que llegue el momento en el que los haya. Por ello, necesitamos conocer el significado de todos los atributos y las relaciones y restricciones existentes entre ellos para establecer las dependencias funcionales y, en definitiva, poder diseñar la base de datos.

Para ilustrar el concepto de dependencia funcional consideremos los atributos `NombreEmp`, `NumSS` y `Direccion`. El conjunto de atributos formado por `NumSS` determina al conjunto de atributos formado por `NombreEmp` y `Direccion`, luego existe una dependencia funcional `NumSS -> {NombreEmp, Direccion}`. Para un `NumSS`, hay una única combinación de valores de `NombreEmp` y `Domicilio`. En cambio, dado un `NombreEmp` no se puede asegurar que exista una única combinación de `NumSS` y `Direccion`, por lo que no existe entre ellos una dependencia funcional.

5.2.2. Reglas de inferencia para dependencias funcionales. Axiomas de Armstrong.

Sea F el conjunto de dependencias funcionales especificadas sobre el esquema de relación R. Generalmente, el diseñador del esquema especifica las dependencias funcionales que son *semánticamente obvias*, pero generalmente hay otras muchas DF que se cumplen. Las dependencias no especificadas pueden *inferirse* a partir de las DF de F. El conjunto de todas las dependencias funcionales de R se denomina *cierre* o *clausura* de F, y se denota con F^+.

Supongamos que especificamos este conjunto de dependencias funcionales obvias sobre el esquema de relación EMPLEADO_DPTO:

F = { `NumSS -> {NombreEmp, FechaEntrada, Direccion, NumDep},`

`NumDep-> {NombreDep, NumSSJefe}}`

Podemos *inferir* estas dependencias funcionales a partir de F:

`NumSS -> {NombreDep, NumSSJefe}`

`NumDep -> NombreDep`

Una DF X -> Y se *infiere* de un conjunto de dependencias F especificado sobre R si X -> Y se cumple en todo estado de relación r que sea una extensión permitida de R. Para inferir las dependencias de forma sistemática, necesitaremos descubrir un conjunto de *reglas de inferencia* que pueda servir para deducir nuevas dependencias a partir de un conjunto dado de dependencias. La notación $F \mid = X$ -> Y indica que la DF X -> Y se infiere del conjunto de DF F. Para denotar grupos de atributos, los concatenaremos sin llaves ni comas. Las tres reglas de inferencia siguientes son los *axiomas de Armstrong*, ya que este demostró que son correctas y completas:

A1 (regla reflexiva): Si $X \supseteq Y$, entonces X -> Y.

A2 (regla de aumento): $\{X$ -> $Y\} \mid = XZ$ -> YZ.

A3 (regla transitiva): $\{X$ -> Y, Y -> $Z\} \mid X$ -> Z.

La regla A1 dice que un conjunto de atributos siempre se determina a sí mismo. Estas dependencias *X -> Y* son *triviales*. En el caso de que *Y* no incluya a *X*, la dependencia funcional es *no trivial*. La regla A2 dice que añadir el mismo conjunto de atributos a los dos miembros de una dependencia produce otra dependencia válida. La regla A3 dice que las dependencias funcionales son transitivas.

Hay otras tres reglas adicionales a los axiomas de Armstrong, que son la regla de descomposición, la de unión y la pseudotransitiva.

5.2.3. Cobertura de un conjunto de dependencias funcionales

Armstrong demostró que las reglas A1, A2 y A3 son correctas y completas. La corrección implica que dado un conjunto de dependencias funcionales *F* especificado sobre un esquema de relación *R*, cualquier dependencia que podamos inferir de *F* con A1, A2 y A3 se cumplirá en todos los estados *r* de *R* que satisfagan las dependencias de *F*. La completitud indica que aplicando A1, A2 y A3 repetidamente para inferir dependencias hasta que no sea posible inferir más, se obtendrá el conjunto completo de *todas* las dependencias posibles que se puedan inferir de *F*. Es decir, el conjunto de dependencias F^+, al que llamamos *cobertura* de *F*, se puede determinar a partir de *F* usando exclusivamente las reglas de inferencia A1, A2 y A3.

Generalmente, al diseñar una base de datos especificamos primero las dependencias funcionales obvias a partir de la semántica de los atributos. Luego podemos usar las reglas A1, A2 y A3 para inferir las dependencias adicionales que también se cumplirán.

Podemos determinar primero todos los conjuntos de atributos *X* que pertenezcan al miembro izquierdo de alguna dependencia funcional, y despues determinar el conjunto de todos los atributos que dependen de *X*. Para cada conjunto de atributos *X*, determinamos el conjunto X^+ de atributos determinados funcionalmente por *X,* que será la cobertura de *X* bajo *F*. Podemos calcular *X* con este algoritmo:

```
X+ := X;
Repetir
   X+_previo := X+;
   Para cada DF Y -> Z en F hacer
     Si X+ ⊇ Y entonces X+ := X+ ∪ Z;
Hasta que (X+_previo = X+);
```

Por ejemplo, consideremos el esquema de la relación EMPLEADO_DPTO; por la semántica de los atributos, especificamos este conjunto de dependencias funcionales:

```
F = { NumSS -> {NombreEmp, FechaEntrada, Direccion, NumDep},
      NumDep-> {NombreDep, NumSSJefe}}
```

Con el algoritmo anterior, podemos calcular los siguientes conjuntos de cierre respecto a *F*:

```
{NumSS}+ = {NumSS, NombreEmp, FechaEntrada, Direccion, NumDep,
      NombreDep, NumSSJefe}
```

$\{NumDep\}^+ = \{NumDep, NombreDep, NumSSJefe\}$

5.2.4. Equivalencia de conjuntos de dependencias funcionales

En esta sección vamos a analizar la equivalencia de dos conjuntos de dependencias funcionales. Primero necesitamos algunas definiciones preliminares. Un conjunto de dependencias funcionales *E* está cubierto por otro conjunto de dependencias funcionales *F* (*F* cubre a *E*) si toda DF de *E* también está en F^+; es decir, *E* está cubierto si toda DF de *E* puede inferirse a partir de *F*. Dos conjuntos de dependencias funcionales *E* y *F* son equivalentes si $E^+ = F^+$, es decir, *E* cubre a *F* y *F* cubre a *E*.

Podemos determinar si *F* cubre a *E* calculando X^+ respecto a *F* para cada DF *X* -> *Y* en *E*, y comprobando después que X^+ incluye los atributos de *Y*, para todas las DF de *E*. Determinaremos si son equivalentes verificando también que *E* cubre a *F*.

5.2.5. Cobertura mínima de dependencias funcionales

Un conjunto de dependencias funcionales es *mínimo* si satisface las siguientes condiciones:

1. Toda dependencia de *F* tiene sólo un atributo en su miembro derecho.
2. No se puede sustituir ninguna dependencia *X* -> *A* en *F* por una dependencia *Y* -> *A*, donde *Y* es un subconjunto propio de *X*, y seguir teniendo un conjunto de dependencias equivalente a *F*.
3. No podemos eliminar ninguna dependencia de *F* y seguir teniendo un conjunto de dependencias equivalente a *F*.

Podemos ver un conjunto mínimo de dependencias como un conjunto de dependencias que está en una forma canónica (sólo un atributo como miembro derecho) y sin redundancias. Una *cobertura mínima* de un conjunto de dependencias funcionales *F* es un conjunto mínimo de dependencias F_{min} que es equivalente a *F*. Desafortunadamente, un conjunto de dependencias funcionales puede tener varias coberturas mínimas. El siguiente algoritmo encuentra una cobertura mínima *G* para cualquier conjunto de dependencias *F*:

1. Asignar $G := F$;
2. Sustituir cada DF $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ en *G* por $X \rightarrow A_1$, $X \rightarrow A_2, \ldots, X \rightarrow A_n$
3. Para cada DF $X \rightarrow A_1$ en *G*
   para cada atributo *B* que sea un elemento de *X*
   si $((G - \{X \rightarrow A\}) \cup \{(X - \{B\}) \rightarrow A\})$ es equivalente a *G*
   entonces reemplazar $X \rightarrow A$ con $(X - \{B\} \rightarrow A$
4. Para cada DF restante $X \rightarrow A$ en *G*
   si $(G - \{X \rightarrow A\})$ es equivalente a *G*
   entonces eliminar $X \rightarrow A$ de *G*

5.3 Normalización

La normalización es un método de diseño de bases de datos relacionales basado en una serie de fundamentos teóricos y con una serie de pasos establecidos: Este método parte del conjunto de todos los atributos o campos a almacenar en la base de datos y obtiene un conjunto de tablas que almacenan toda la información (datos y relaciones) de la base de datos. Esto evita una serie de problemas y anomalías que pueden derivarse de un diseño erróneo en el que no se haya realizado una verificación formal. El proceso de normalización es un proceso descendente, que consiste en analizar y descomponer las relaciones que forman el esquema de una base de datos de forma que satisfagan una serie de características, dando lugar a mejores agrupaciones de atributos para la formación de tablas o relaciones.

5.3.1 Introducción a la normalización

Vamos a estudiar inicialmente las formas normales básicas propuestas por Codd para el diseño de bases de datos relacionales, y que son la 1FN, 2FN y 3FN. Todas estas formas normales se basan en el estudio de las dependencias funcionales existentes entre los atributos de una relación. Los esquemas de relación que no cumplan ciertos criterios, conocidos como las pruebas de las formas normales, deberán ser descompuestos en esquemas de relación más pequeños que satisfagan dichos criterios, obteniendo al final un conjunto de tablas que minimizan la redundancia y las anomalías de actualización descritas anteriormente.

En el proceso de normalización se debe cumplir que al aplicar una forma normal determinada, se produce descomposición sin pérdida y se conservan las dependencias. Es decir, las tablas que resulten de la descomposición de una tabla que no cumple los criterios de una forma normal dada conservan las dependencias funcionales existentes en la relación original, y, además, si se realiza el producto natural de las nuevas tablas obtenidas se obtiene exactamente la tabla original.

Antes de pasar a estudiar las formas normales, recordemos los conceptos de superclave, clave candidata y clave primaria. Una superclave de un esquema de relación $R = \{A_1, A_2, ..., A_n\}$ es un conjunto de atributos $S \subseteq R$ que cumple que dadas dos tuplas distintas cualesquiera t_1 y t_2 de R se cumple que $t_1[S] \neq t_2[S]$. Una superclave K se dice que es una clave candidata si es mínima, es decir, si al quitarle cualquier atributo deja de ser superclave. La clave candidata elegida para ser la clave de un esquema de relación se denomina clave primaria. Por último, y en cuanto a la denominación de atributos distinguiremos entre atributos primos y atributos no primos. Un atributo de una relación es primo si es miembro de alguna clave candidata de la relación, y es no-primo si no lo es.

5.3.2. Primera forma normal (1FN)

La primera forma normal dice que los dominios de los atributos sólo deben incluir valores individuales (indivisibles), y que el valor de cualquier atributo de una tupla debe ser un único valor perteneciente a ese dominio. Por tanto, sólo se permiten **valores**

atómicos para los atributos, y quedan prohibidas las **relaciones anidadas** y los **atributos multivaluados**. Actualmente, forma parte de la definición formal de relación en el modelo relacional básico, y se elimina en el modelo relacional anidado y en los sistemas objeto-relacionales.

La definición formal es: Una relación está en primera forma normal si sus atributos no tienen dominios que a su vez sean conjuntos.

Vamos a usar un ejemplo para ilustrar el proceso de normalización. Supongamos que tenemos información sobre pedidos, con los siguientes atributos:

`Num_Pedido, Fecha, Num_Proveedor, Nombre_Proveedor, Dirección_Proveedor, Num_Producto, Descripción, Precio, Cantidad, Precio Total_Producto, Precio_Total_Pedido`

Considerando todos estos atributos como una relación universal que contiene toda la base de datos, hay que buscar una clave para dicha relación. En este caso, tenemos información de pedidos, con datos únicos para cada pedido y datos que se repiten dentro de un pedido. Parece lógico considerar que es una tabla de pedidos. Como para cada pedido hay un `Num_Pedido` único, ésta será nuestra clave inicial.

Una vez elegida la clave inicial, tenemos que ver si todos los atributos restantes tienen dominios que no sean conjuntos.

Los atributos `Fecha`, `Num_Proveedor`, `Nombre_Proveedor`, `Direccion_Proveedor` y `Precio_Total_Pedido` son únicos para cada pedido.

Los atributos `Num_Producto`, `Descripcion`, `Precio`, `Cantidad` y `Precio_Total_Producto` no son únicos para cada pedido, sino que para cada pedido tendrán tantos valores como líneas de productos haya incluidas en el pedido. Por lo tanto, la relación `Pedidos` que incluye a todos los atributos no está en 1FN, y es necesario realizar una transformación para obtener relaciones que estén en 1FN.

Tenemos dos métodos alternativos:

a) Descomponer la relación que no cumple la 1FN en dos, pasando los atributos multivaluados a una nueva tabla, junto con la clave respecto a la que se repiten.

b) Añadir a la clave de la relación que no cumple la 1FN los atributos necesarios para identificar realmente cada fila de la tabla, considerando que los atributos ya no son multivaluados.

En principio, es más aconsejable usar el primer método, puesto que avanza en el sentido de ir descomponiendo la base de datos en tablas.

Aplicando el caso a nuestro ejemplo, con el método a) tendríamos:

`Pedidos (Num Pedido, Fecha, Num_Proveedor, Nombre_Proveedor, Direccion_Proveedor, Precio_Total_Pedido)`

`Lineas (Num Pedido, Num Producto, Descripcion, Precio, Cantidad, Precio Total_Producto)`

Observemos que la clave primaria de `Lineas` está compuesta por el número de pedido y el número de producto, porque se supone que en un pedido no aparece dos veces el mismo producto.

Con el método b), no se descompondría la tabla, sino que simplemente su clave primaria sería `Num_Pedido` y `Num_Producto`, y los atributos ya no serían multivaluados.

En cualquiera de ambos casos, las relaciones obtenidas sí cumplen la 1FN.

Una relación puede tener varios grupos de atributos multivaluados, e incluso grupos anidados, que habrá que ir descomponiendo por niveles (lo veremos en otros ejercicios).

5.3.3. Segunda forma normal (2FN)

Una vez que tenemos un esquema de relaciones en 1FN, observamos que seguimos teniendo posibilidad de anomalías. En nuestro ejemplo, tenemos repetida la información de proveedores y productos para cada pedido, y no se pueden introducir productos sin pedido.

Como siguiente nivel de verificación, vamos a ver la segunda forma normal, que busca suprimir las dependencias funcionales de partes de la clave primaria. Así, la segunda forma normal está basada en el concepto de *dependencia funcional total*.

La definición formal es: Una relación está en segunda forma normal si está en 1FN y todos sus atributos no primos (que no pertenecen a ninguna clave candidata) dependen funcionalmente de la clave primaria completa, y no de parte de ella.

Volviendo a nuestro ejemplo, la relación `Pedidos` está en 2FN, puesto que no hay claves candidatas distintas de la primaria, y mirando los atributos de la clave primaria, vemos que solo tiene un atributo, por lo que no puede haber dependencias parciales.

En cambio, la relación `Lineas` no está en 2FN: tampoco hay claves candidatas distintas de la primaria, y vemos que para un número de producto, hay una única descripción y precio del producto, luego `Descripcion` y `Precio` dependen funcionalmente sólo de `Num_Producto`, que es una parte de la clave primaria de la tabla.

La transformación a realizar es la siguiente: Llevar los atributos no primos que dependen de parte de la clave primaria a una nueva tabla, y añadirles la parte de la clave primaria de la que dependen.

En nuestro caso, nos llevaremos `Descripcion` y `Precio` de la tabla `Lineas`, y les añadiremos `Num_Producto`, creando una nueva tabla llamada `Productos`:

`Lineas (Num Pedido, Num Producto, Cantidad, Precio_Total_Producto)`

`Productos (Num Producto, Descripción, Precio)`

Por tanto, el esquema de nuestra base de datos consta ya de tres relaciones: `Pedidos`, `Lineas` y `Productos`.

En otros casos, puede haber varios atributos o grupos de ellos que dependan de partes distintas de la clave primaria, creándose una tabla para cada grupo correspondiente a una parte de la clave primaria. No hay casos de tener que aplicar esta transformación de forma anidada.

5.3.4. Tercera forma normal (3FN)

La tercera forma normal está basada en el concepto de dependencia funcional transitiva. Una dependencia funcional *X -> Y* entre dos conjuntos de atributos *X* e *Y* es una *dependencia funcional transitiva* si existe un tercer conjunto de atributos *Z* disjunto de ellos para el que existe una dependencia funcional *X -> Z y Z -> Y*. En una relación en tercera forma normal, ningún atributo no primo puede depender transitivamente de la clave primaria. Para descomponer una relación que no está en 3FN en relaciones que sí lo estén creamos una nueva relación con los atributos no primos que dependen transitivamente de la clave primaria junto con los atributos de los que dependen funcionalmente de forma directa. Además, en el esquema original se eliminan todos los atributos que tienen dependencia transitiva respecto a la clave primaria.

Aplicado esto al ejemplo que estamos siguiendo en este tema, vemos que la relación Pedidos no está en tercera forma normal, ya que existe una dependencia funcional transitiva entre `Nombre_Proveedor`, `Dirección_Proveedor` y `Num_Pedido` a través de `Num_Proveedor`, ya que existe una dependencia funcional `{Num_Pedido} -> {Num_Proveedor}` y otra `{Num_Proveedor} -> {Nombre_Proveedor, Dirección_Proveedor}`. Así, el esquema `Pedidos` se transforma en los esquemas `Pedidos` y `Proveedores` siguientes.

`Pedidos (Num_Pedido, Fecha, Num_Proveedor, Precio_Total_Pedido)`

`Proveedores (Num_Proveedor, Nombre_Proveedor, Direccion_Proveedor)`

A continuación se muestra de forma resumida qué comprueba cada forma normal y qué proceso se debe llevar a cabo para hacer que un esquema esté en una forma normal determinada.

Primera forma normal (1FN): Una relación no debe contener atributos no atómicos o multivaluados ni debe contener relaciones anidadas. Si una relación no cumple esta condición, es decir, no está en 1FN, se crean nuevas relaciones con los atributos multivaluados o relaciones anidadas.

Segunda forma normal (2FN): Una relación que tiene una clave primaria compuesta no debe contener atributos no primos que dependan funcionalmente de parte de la clave. Por tanto, todas las relaciones con claves simples están en 2FN. Su una relación no está en 2FN se crea una nueva relación con cada clave parcial y el conjunto de atributos que dependen funcionalmente de ella. No obstante, hay que tener en cuenta

que debe seguir existiendo una relación con la clave primaria original y los atributos que dependen totalmente de ella.

Tercera forma normal (3FN): Una relación no debe tener atributos no primos que dependan funcionalmente de atributos no primos, es decir, no deberían existir dependencias funcionales transitivas por parte de los atributos no primos. Si una relación no está en 3FN se crea una nueva relación con los atributos no primos que determinan funcionalmente los otros atributos no primos.

Las definiciones de 2FN y 3FN pueden generalizarse para tener en cuenta las dependencias parciales o transitivas de *todas* las claves de un esquema de relación.

5.3.5. Forma normal de Boyce-Codd (FNBC)

La FNBC es más estricta que la 3FN, y lo que nos dice es que todo determinante de una dependencia funcional no trivial de un esquema de relación tiene que ser clave candidata.

Consideremos este ejemplo. Tenemos la relación `IMPARTE`, cuyo esquema es (`Alumno`, `Asignatura`, `Profesor`). Cada `Profesor` imparte una sóla asignatura, por lo que hay una DF `Profesor -> Asignatura`. Esta relación está en 1FN, 2FN y 3FN, pero es claramente redundante, ya que decimos quién es el profesor de cada asignatura para cada alumno. No está en FNBC, ya que hay un determinante (`Profesor`) que no es clave candidata.

La descomposición correcta en dos tablas es la que lleva a una nueva tabla el atributo del miembro izquierdo de la dependencia funcional, y le añade el miembro derecho. En nuestro ejemplo, crearíamos la relación `PROFESOR(Profesor, Asignatura)`, y la relación original quedaría como `IMPARTE(Alumno, Profesor)`.

5.3.6. Otras formas normales

En esta sección presentaremos el concepto de *dependencia multivaluada*, en el que se basa la 4FN, y el concepto de *dependencia de producto*, en el que se basa la 5FN.

Una *dependencia multivaluada* (DMV) X ->> Y, especificada en el esquema de relación R, impone la siguiente restricción sobre cualquier estado r de R: si existen dos tuplas t_1 y t_2 en r tales que $t_1[X] = t_2[X]$, entonces deberán existir también dos tuplas t_3 y t_4 que verifiquen lo siguiente, donde Z representa el resto de los atributos de R:

- $t_1[X] = t_2[X] = t_3[X] = t_4[X]$
- $t_3[Y] = t_1[Y]$ y $t_4[Y] = t_2[Y]$
- $t_3[Z] = t_2[Z]$ y $t_4[Z] = t_1[Z]$

Si X ->> Y, entonces también se cumple que X ->> Z, lo que se denota como X ->> $Y \mid Z$.

Podemos ver un ejemplo con esta relación `BIBLIOGRAFIA(Libro,Asignatura,Profesor)`, que contiene los libros

recomendados por los profesores que imparten asignaturas. Resulta que todos los profesores que imparten una asignatura recomiendan el mismo conjunto de libros, un libro puede ser recomendado para varias asignaturas, y un profesor puede impartir varias asignaturas. Por lo tanto, si se asigna un nuevo profesor a una asignatura, hay que insertar tantas filas como libros recomendados para ella, y si se recomienda un nuevo libro, hay que insertar tantas filas como profesores de la asignatura.

Esta relación presenta dependencias multivaluadas, ya que dada una asignatura, aparece la combinación de todos los libros recomendados con todos los profesores.

Formalmente, diremos que una relación está en 4FN si para cada dependencia multivaluada no trivial, el determinante de la misma es clave candidata.

Para resolver el problema, podemos descomponer esta relación en dos:

LIBROS(Libro,Asignatura)

PROFESORES(Asignatura,Profesor)

La primera relación contiene los atributos implicados en los dos miembros de la dependencia multivaluada, y la segunda los atributos originales eliminando la parte derecha de la dependencia multivaluada. Esta descomposición es una descomposición sin pérdida, ya que si combinamos ambas tablas por su atributo común, recuperamos la información adicional, sin tuplas incorrectas.

Existen situaciones en las que una relación puede no presentar dependencias multivaluadas y seguir teniendo redundancia evitable. Este es el caso cuando existen dependencias de producto o reunión. Un esquema de relación tiene una *dependencia de producto* cuando es posible descomponerla sin pérdida en más de dos relaciones, pero no en dos.

Supongamos que tenemos la misma relación BIBLIOGRAFIA del caso anterior, pero que en este caso cada profesor recomienda libros distintos en la misma asignatura. En este caso, no existe una dependencia multivaluada, ya que si hacemos la descomposición y luego la combinación, tendremos filas incorrectas. Pero existe redundancia, ya que vestamos repitiendo cada asignatura impartida por cada profesor para cada libro recomendado. Este es el tipo de redundancia que intenta evitar la 5FN.

Decimos que una relación está en 5FN está en 4FN y para cada dependencia de producto no trivial, todas las relaciones en las que se descompone son claves candidatas de la propia relación.

En el ejemplo anterior, BIBLIOGRAFIA presenta una dependencia de producto con tres relaciones, y ninguna de ellas es clave candidata, luego no está en 5FN. Para llegar a 5FN, necesitamos descomponer BIBLIOGRAFIA en tres relaciones:

LIBROS(Libro,Asignatura)

PROFESORES(Asignatura,Profesor)

RECOMENDADOS(Libro,Profesor)

Al combinar estas tres relaciones, obtendremos la relación original.

5.4. Proceso general de diseño de bases de datos relacionales

5.4.1. El modelo Entidad-Relación y la normalización

5.4.2. Desnormalización para el rendimiento