Tema 6: Teoría de la Normalización

1. Introducción

Si definimos una base de datos como; "*una colección de información estructurada, referente a objetos y hechos de la realidad, y almacenados en un ordenador de forma que su acceso, mantenimiento y recuperación sean eficaces*", cuando diseñemos una base de datos relacional, en general pretenderemos obtener un conjunto de esquemas de relación que nos permitan almacenar la información sin redundancias y de forma que podamos recuperar dicha información fácilmente. El problema que se nos plantea es decidir cuál es la estructura lógica adecuada para el conjunto de datos para los cuales queremos representar la base de datos, es decir, el decidir cuáles deben ser las relaciones que deberían existir y qué atributos deberán contener.

Hasta ahora hemos aprendido ha diseñar un modelo conceptual de la base de datos, modelo entidad relación, y establecer su diseño lógico, modelo relacional, obteniendo así una serie de tablas finales que son las candidatas a formar nuestra base de datos. Sin embargo, dichas tablas han sido obtenidas a partir de un diseño conceptual elaborado sin ningún tipo de reglas, por lo que podemos obtener un diseño de tablas más o menos heterogéneo.

En el siguiente punto veremos la teoría de la normalización que consiste en un conjunto de reglas formales que nos permiten confeccionar un diseño lógico, o asegurar que un diseño lógico que ya hemos confeccionado cumpla una serie de propiedades, corrigiendo la estructura de los datos de las tablas y evitando una serie de problemas tales como:

- Incapacidad de almacenar ciertos hechos.
- Redundancias y, por tanto, posibilidad de inconsistencias.
- Ambigüedades.
- Pérdida de información.
- Aparición en la B.D. de estados no válidos en el mundo real, es lo que se llama anomalías de inserción, borrado y modificación.

En general, el objetivo del diseño lógico es convertir un esquema conceptual en un esquema lógico que se ajuste al sistema de gestión de base de datos a utilizar. Por tanto, se pretende que el esquema lógico cumpla ciertas características:

- Las relaciones deben de estar en tercera forma normal.
- Se deben definir las claves primarias y ajenas de todas las relaciones.
- Es necesario incluir en el esquema las reglas de integridad necesarias.

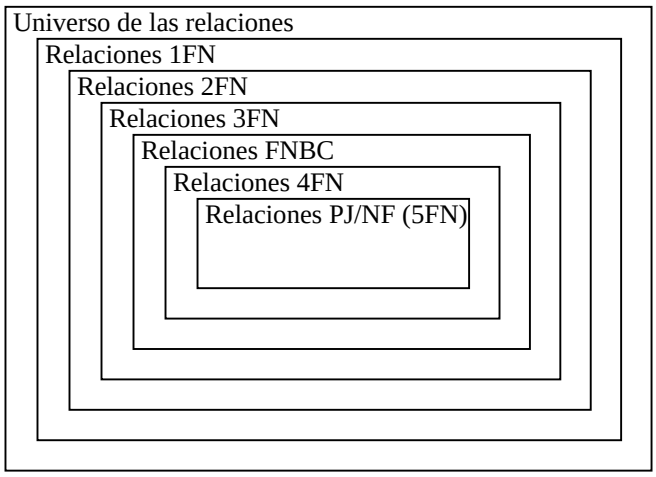
2. Teoría de la normalización

La teoría de la normalización, desarrollada por Codd en 1972 permite mejorar el diseño lógico de un sistema de información. Las reglas formales en las que se fundamenta la teoría de la normalización son conocidas con el nombre de Formas Normales, que son un conjunto de restricciones que deben de cumplir las relaciones.

Existen seis formas normales, de forma que cuando la base de datos cumple las reglas de la primera forma normal se considera que está en primera forma normal (1FN), cuando pasan la segunda, que está en segunda forma normal (2FN), etc. Además, una base de datos de la que se afirme que está en 2FN, está también en 1FN, pues las formas normales se aplican de forma sucesiva.

En principio, Codd definió la 1FN, 2FN y 3FN, con la idea de que era más deseable que una relación estuviese en 2FN que en 1FN, y a su vez, era mejor que estuviese en 3FN que en 2FN. También se introdujo la idea de un procedimiento, el llamado *procedimiento de normalización*, con el cual una cierta relación en una determinada FN puede convertirse en un conjunto de relaciones más deseables (o sea, en una FN superior). Además, este procedimiento es reversible, lo que garantiza que no se pierde información en cada paso del proceso.

Con posterioridad (1974), y dado que la 3FN no contemplaba algunos casos particulares, Boyce ayudó a Codd a redefinir la 3FN y fue lo que se llamó la forma normal de BOYCE/CODD (FNBC). En 1977 y 1979, Fagin introdujo la 4FN y 5FN respectivamente. En la figura se muestra como se pueden agrupar las relaciones en función de su estado de normalización.



De las seis formas normales, generalmente solo se aplican sobre las bases de datos las tres primeras y/o la de Boyce/Codd, considerando que una base de datos que está en 3FN o en FNBC es una base de datos correctamente diseñada. Por ello, en este curso solo expondremos estás tres primeras formas normales.

Las formas normales se basan en el concepto de *dependencia*, que comprende las restricciones definidas sobre los atributos de una relación. Tenemos tres tipos de dependencias:

- Dependencias funcionales. Se usa para la 1FN, 2FN, 3FN y FNBC.
- Dependencias multivaluadas (4FN)
- Dependencias de Join o de combinación (5FN)

Dado que vamos a normalizar solo hasta la 3FN, veremos únicamente en qué consiste la dependencia funcional.

2.1. 1FN, 2FN y 3FN

Se define el concepto de **dependencia funcional** como:

Dada una relación R, el atributo Y de R depende funcionalmente del atributo X de R (R.X → R.Y) si y sólo si un valor de Y en R está asociado a cada valor X en R (en cualquier momento dado).

Obviamente, si el atributo X es una clave candidata de la relación R (o es la clave primaria), entonces todos los atributos Y de la relación deben depender funcionalmente de ese atributo X.

Otra manera más clara de expresar el significado de una dependencia funcional es:

Dada una relación R, el atributo Y de R depende funcionalmente del atributo X de R si y sólo si siempre que dos tuplas de R concuerden en su valor de X, deben por fuerza concordar en su valor de Y.

Definimos también el concepto de **dependencia funcional completa** como:

Se dice que el atributo Y de la relación R es por completo dependiente funcionalmente del atributo X de R si depende funcionalmente de X y no depende funcionalmente de ningún subconjunto propio de X.

Después de introducir estos conceptos ya podemos definir las tres primeras formas normales. Por simplicidad, vamos a definirlas para el caso en el que las relaciones sólo tienen una clave candidata (y por tanto, solo tienen clave primaria). Para el caso en que esto no se cumpla, se recurre directamente a la FNBC.

- **1FN**. Una relación está en primera forma normal si y sólo si satisface que sus dominios simples sólo tienen valores atómicos, es decir, si todos sus atributos son atómicos.

- **2FN**. Una relación se encuentra en segunda forma normal si y sólo si está en primera forma normal y todos los *atributos no clave* (*) dependen por completo de la clave primaria.

- **3FN**. Una relación está en tercera forma normal si y sólo si está en segunda forma normal y además, cada *atributo no clave* (*) depende de la clave primaria de modo no transitivo. Dicho de otra forma, una relación está en tercera forma normal si y sólo si sus atributos no clave son:
  - Mutuamente independientes, es decir, no existe un atributo no clave que dependa funcionalmente de alguna combinación del resto de atributos no clave.
  - Por completo dependientes funcionalmente de la clave primaria.

2.2. FNBC

El problema de la 3FN es que no maneja relaciones que:

- Tienen varias claves candidatas,
- Esas claves candidatas son compuestas, y
- Las claves candidatas tienen por lo menos un atributo en común.

Por ello, se define la FNBC para el caso en el que exista más de una clave candidata, y que se cumplan las condiciones anteriores. En el caso en el que no se den dichas condiciones, o bien no exista más de una clave candidata (solo la clave primaria) la FNBC es completamente equivalente a la 3FN. La FNBC es más restrictiva que la 3FN ya que solo permite que un atributo no clave dependa funcionalmente de la clave primaria.

3. Metodología

En este punto vamos a ver, de forma práctica, cómo hacer uso de la teoría de normalización de una base de datos. Para ello, vamos a ir aplicando cada una de las formas normales sobre el ejemplo práctico en que se nos pidieran diseñar una base de datos para la parte de gestión de una empresa correspondiente a la facturación de los clientes.

La primera fase de diseño de una base de datos correspondía a la especificación de requisitos de usuario. Por lo tanto, después de realizado un análisis, el diseño que presentan las facturas es el mostrado en la figura.

Para identificar la factura, hemos considerado como clave primaria el código de la factura y además, hemos deducido que necesariamente una factura debe poseer todos esos campos.

Este es el diseño inicial de las FACTURAS, al cual debemos aplicarle la metodología de las formas normales para ver si se trata de un buen diseño (aunque dados nuestros conocimientos previos podemos afirmar que no lo es).

FACTURA

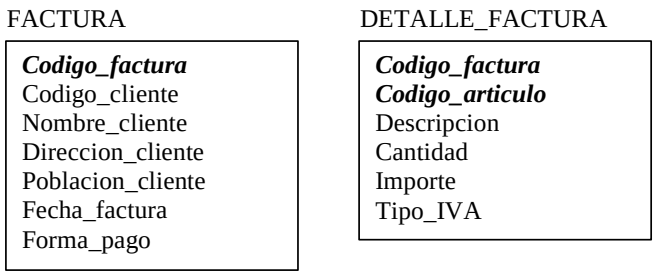
Codigo_factura
Codigo_cliente
Nombre_cliente
Direccion_cliente
Poblacion_cliente
Fecha_factura
Forma_pago
Codigo_articulo_1
Descripcion_1
Cantidad_1
Importe_1
Tipo_IVA_1
…
Codigo_articulo_N
Descripcion_N
Cantidad_N
Importe_N
Tipo_IVA_N

(*) Atributo no clave significa que no participa en la clave primaria.

3.1. Primera forma normal (1FN)

Recordemos que una base de datos se considera que está en 1FN si cada atributo (campo) de una tabla contiene un solo valor atómico (simple). Un atributo que contiene varios valores puede derivar en una perdida de datos y por lo tanto no nos interesa.

Analizando el diseño inicial de la tabla FACTURA, observamos la existencia de múltiples valores para los atributos siguientes: *Codigo_articulo, descripcion, cantidad, importe e IVA*. Por lo tanto, observamos que no cumple la condición de 1FN. La solución consiste en crear una nueva tabla, que podemos llamar DETALLE_FACTURA, a la cual se trasladan los datos repetitivos, en nuestro caso los datos referentes a los artículos (codigo_articulo, descripción, cantidad, importe e IVA). Aplicando esto, el diseño de la base de datos para las facturas en 1FN sería:



Como regla a aplicar, cuando se produce la separación de datos de la tabla original en una nueva tabla ésta, además de los atributos necesarios, traslada la clave primaria de la tabla original como parte de su nueva clave primaria, y por tanto estará formada generalmente por dos atributos.

3.2. Segunda forma normal (2FN)

La segunda forma normal, como la tercera que veremos a continuación, se relaciona con el concepto de dependencia funcional.

Entendemos como dependencia funcional a la relación que tienen los atributos (campos) de una tabla con otros atributos de la propia tabla. Un campo tiene dependencia funcional si necesita información de otro/s campo/s para poder contener un valor.

Una tabla se dice que esta en segunda forma normal (2FN) si sucede que:

- Está en 1FN
- Cada atributo (campo) no clave depende de la clave completa, no de parte de ella.

Por supuesto, una base de datos estará en 2FN si todas sus tablas lo están.

La idea intuitiva de la 2FN es identificar todas las tablas con una clave compuesta, pues todas las tablas con clave simple están por defecto en 2FN si están en 1FN, y comprobar que cada uno de los campos de esta tabla depende de la clave completa.

En nuestro ejemplo, la tabla FACTURA se encuentra en 2FN pues está en 1FN y su clave es simple. Sin embargo, la tabla DETALLE_FACTURA ha de ser analizada pues su clave es compuesta (esta formada por dos atributos).

Analizando la tabla DETALLE_FACTURA, observamos que el atributo *descripcion* depende únicamente del atributo *codigo_articulo* (la descripción de un artículo depende únicamente de qué artículo se trate y es completamente independiente de la factura), por lo cual la *descripcion* habrá de ser llevada a una nueva tabla junto con el atributo clave *codigo_articulo*.

Supongamos además que el cliente nos indica que en la factura, aunque cada artículo posee calculado su IVA, el tipo de IVA que aplica es común a toda la factura y no depende en cada factura de los artículos. En este caso, el atributo *Tipo_IVA* solo dependerá funcionalmente del *codigo_factura* y no depende de *codigo_articulo*, por lo cual ha de ser devuelto a la tabla FACTURA como un único atributo *Tipo_IVA* que depende solo de la clave de FACTURA (*codigo_factura*). Con estas consideraciones, el diseño de la base de datos para las facturas de la empresa expresado en 2FN sería:

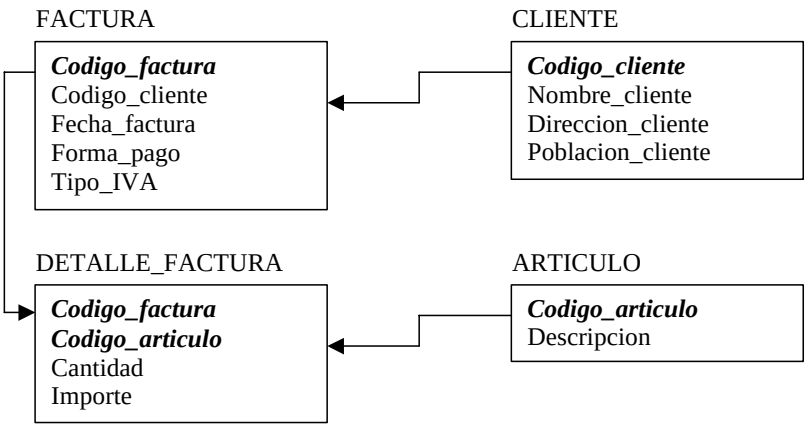
3.3. Tercera forma normal (3FN)

Recordemos que una tabla se dice que está en tercera forma normal (3FN) si:

- Está en 2FN.
- Todos los atributos que no son claves deben ser mutuamente independientes, es decir, un atributo no debe depender de otro atributo no clave de su tabla.

Si un atributo que no es clave depende de otro atributo que no es clave, la tabla posiblemente contiene datos acerca de más de una entidad, contradiciendo el principio de que cada tabla almacene información de una entidad.

En nuestro ejemplo, podemos observar que las tablas ARTICULO y DETALLE_FACTURA se encuentran en 3FN. Sin embargo, la tabla FACTURA no está en 3FN, pues los atributos *Nombre_cliente, Direccion_cliente* y *Poblacion_cliente* dependen funcionalmente del atributo *Codigo_cliente*, campo que no es clave. Por ello, debemos extraer estos atributos de la tabla FACTURA e incluirlos en una nueva tabla que haga referencia al cliente, tabla que llamaremos CLIENTE y que contendrá como clave primaria el *Codigo_cliente* y como atributos el *Nombre_cliente, Direccion_cliente* y *Poblacion_cliente*. Aplicando esto, nuestro diseño de la base de datos para las facturas da lugar a las tablas que pueden verse en la siguiente figura y que ya están en 3FN por lo que podemos considerar que es un buen diseño.



Como detalle, hay que destacar que la última descomposición se podría haber realizado de la forma:
Codigo_factura→ Codigo_cliente , Codigo_factura→ Nombre_cliente , …

Pero es evidente que con esta descomposición se pierde información sobre la dependencia Codigo_cliente → Nombre_cliente y por tanto sería una mala descomposición. Entonces, en las relaciones con transitividad habrá que obrar con cautela antes de decidir que tipo de descomposición se realiza.

3.4. Consideraciones finales y problemas de la normalización

La teoría de la normalización nos ayuda a estructurar mejor las tablas de la base de datos, evitando posibles redundancias.

Por otra parte, si seguimos la metodología de diseñar primero el modelo E-R y obtener así un diseño conceptual que después es convertido en diseño lógico, modelo relacional, este diseño lógico resultante estará en 3FN siempre que todo el proceso se haya realizado de forma correcta, sirviendo en este caso la teoría de la normalización para comprobar que el diseño ha sido realizado correctamente. Si no lo fuese, podremos aplicar las formas normales para corregir los errores que hubieran podido producirse.

Mientras la normalización resuelve los problemas relacionados con la estructuración de los datos en tablas, crea problemas añadidos a su propio concepto, como son la duplicación de datos y la ineficacia en la recuperación de información.

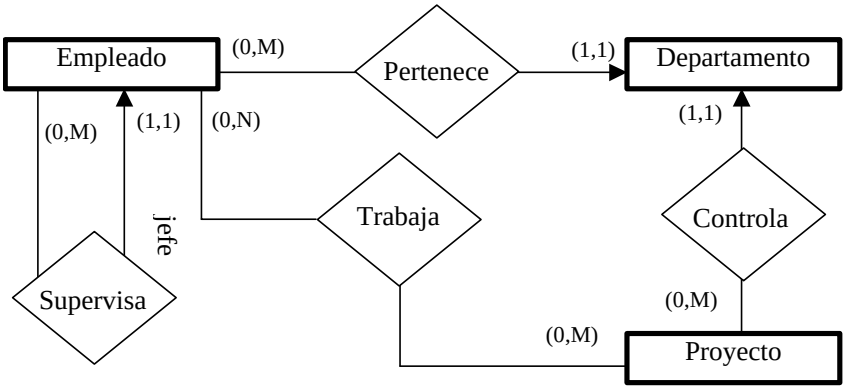
Así, el proceso de normalización envuelve la descomposición de una tabla en tablas más pequeñas, lo cual requiere que la clave primaria de la tabla original se incluya, como una clave ajena, en la tabla/s que se forman. Esto significa que a medida que se van creando estas claves ajenas se va incrementando las probabilidades de poner en peligro la integridad de la base de datos. Esta es la razón de que, normalmente, solo se normalice hasta la 3FN.

Otro efecto adicional del número creciente de tablas en la base de datos, es que se ve disminuido el rendimiento del sistema en la recuperación de la información contenida. Esta disminución del rendimiento puede ser particularmente importante en sistemas basados en microordenadores. Por tanto, en ciertas ocasiones es necesario llegar a un compromiso entre el nivel de normalización de la base de datos y el rendimiento del sistema.

4. Ejemplos de diseño de bases de datos

4.1 Ejemplo 1: Diseñar el modelo E-R y aplicar normalización

Una empresa pretende desarrollar una base de datos de empleados y proyectos. La empresa esta estructurada en departamentos, cada uno de los cuales posee uno o varios proyectos, de forma que un proyecto solo depende de un departamento. Por otro lado, cada departamento consta de uno o varios empleados que trabajan de forma exclusiva para ese departamento, pero pueden trabajar simultáneamente en varios proyectos. Cada empleado tiene un jefe encargado de supervisar su trabajo, pudiendo cada jefe supervisar el trabajo de varios empleados. Dada la descripción anterior, desarrollar la base de datos normalizada hasta 3FN.



Este es el diseño conceptual que hemos desarrollado para solucionar el problema.

Una vez realizado el esquema conceptual, pasamos a realizar el diseño lógico, modelo relacional. Para ello aplicamos las tres reglas generales de forma sucesiva:

De la regla referente a las entidades, obtenemos la existencia de las tres tablas siguientes:

EMPLEADO

Codigo_empleado
Nombre
Edad
Direccion
Poblacion

DEPARTAMENTO

Codigo_dpto
Nombre

PROYECTO

Codigo_proyecto
Descripcion

Aplicando la regla referente a las relaciones 1:M, obtenemos la inclusión de tres claves ajenas en las tablas, dos en la tabla empleado y una en la tabla proyecto, quedando las tablas de la siguiente forma:

EMPLEADO

Codigo_empleado
Nombre
Edad
Direccion
Poblacion
Codigo_director
Codigo_dpto

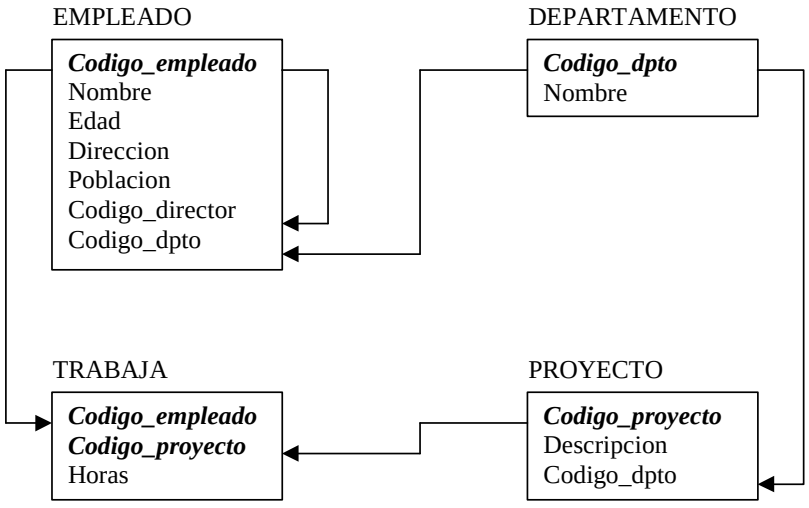
DEPARTAMENTO

Codigo_dpto
Nombre

PROYECTO

Codigo_proyecto
Descripcion
Codigo_dpto

Aplicando ahora la regla referente a las relaciones M:N, creamos una nueva tabla llamada TRABAJA que relaciona el empleado con los proyectos en los que trabaja y viceversa, quedando las tablas como:



Se pueden comprobar que el diseño obtenido cumplen las tres formas normales.

4.2 Ejemplo 2: Aplicación de las FN

Supongamos que nos plantean desarrollar una B.D. para una empresa de transporte que quiere gestionar los envíos de pedidos a los clientes. Podemos plantearnos un diseño inicial de una tabla para esta base de datos que contenga la información del código del envío, el camión que lo transporta, los datos del cliente al que va dirigido así como cada uno de los artículos que componen el envío incluyendo sus características, tal y como se muestra en la figura siguiente:

Es evidente que este diseño inicial no es bueno, por lo que habrá que aplicar las tres primeras formas normales y llevar el diseño a 3FN.

ENVIO

Codigo_envio
Matricula_camion
Modelo_camion
Capacidad_camion
Cliente_1
Direccion_cliente_1
Pedido_cliente_1
Articulo_1_pedido_cliente_1
Volumen_articulo_1_pedido_cliente_1
…
Articulo_I_pedido_cliente_1
Volumen_articulo_I_pedido_cliente_1
…
Cliente_N
Direccion_cliente_N
Pedido_cliente_N
Articulo_1_pedido_cliente_N
Volumen_articulo_1_pedido_cliente_N
…
Articulo_J_pedido_cliente_N
Volumen articulo J pedido cliente N

La 1FN debemos aplicarla a la tabla ENVIO, obteniendo una nueva tabla PEDIDO, sobre la cual debemos volver a aplicar la 1FN, obteniendo una nueva tabla PEDIDO_ARTICULO, para poder tener valores atómicos. Después de esto, el esquema queda como sigue:

ENVIO

Codigo_envio
Matricula_camion
Modelo_camion
Capacidad_camion

PEDIDO_CLIENTE

Pedido_cliente
Cliente
Direccion_cliente
Codigo_envio

PEDIDO_ARTICULO

Pedido_cliente
Codigo_articulo
Volumen

Apliquemos ahora la 2FN; observamos que en la tabla PEDIDO_ARTICULO el campo *volumen* solo depende del *codigo_articulo*, no dependiendo para nada del *pedido_cliente*, por lo cual obtenemos una nueva tabla, que llamaremos ARTICULO:

ENVIO

Codigo_envio
Matricula_camion
Modelo_camion
Capacidad_camion

PEDIDO_CLIENTE

Pedido_cliente
Cliente
Direccion_cliente
Codigo_envio

PEDIDO_ARTICULO

Pedido_cliente
Codigo_articulo

ARTICULO

Codigo articulo
Volumen

Aplicando la 3FN, vemos que en la tabla ENVIO los datos *modelo_camion* y *capacidad_camion* dependen de *matricula_camion*, que no es clave primaria, por lo cual debemos sacarlos en una nueva tabla. Esto mismo sucede con el dato *direccion_cliente* respecto a *cliente* en la tabla PEDIDO_CLIENTE. Teniendo esto en cuenta, el diseño final queda como: