Caso práctico

Víctor *ha aprendido mucho sobre bases de datos, y no tiene demasiados problemas a la hora de hacer los diagramas E-R o el esquema relacional para las bases de datos que van a usar en las distintas aplicaciones que la empresa* ***SI Andalucía*** *está desarrollando. No obstante, los diseños que él hace son mejorables en algunos casos, debido a que es posible* ***reducir el grado de redundancia*** *de la información, y evitar problemas en inserciones, modificaciones y borrados, evitando* ***inconsistencias de la información*** *almacenada y permitiendo almacenar toda la información que se necesita. A veces los diseños de* ***Víctor*** *no son los mejores para conseguir estos fines, y* ***José*** *sabe la causa: Todavía no ha aprendido nada sobre normalización de bases de datos.*

Por eso se reúne con ***Víctor*** *y* ***María*** *para tratar este tema: Ha llegado la hora de enseñar a* ***Víctor*** *la teoría y la práctica de la* ***normalización*** *de bases de datos, para que pueda implicarse en proyectos más ambiciosos, encargándose enteramente del diseño de la base de datos, cumpliendo con todos los requisitos de calidad necesarios.*

Al principio, ***Víctor*** *no entiende qué es lo que falla en los diseños que él ha hecho, y* ***José*** *le muestra como ejemplo el diseño de una base de datos de la consulta de una clínica veterinaria, en la que él había incluido una única relación:*

`MASCOTA(cod_mascota, nombre, raza, direccion, telefono, enfermedad, color, peso, fecha_visita, tratamiento)`

Le hace ver que en realidad, aunque toda esa información se refiere a la mascota, hay dos hechos distintos:

- *Por un lado los datos de la mascota propiamente dichos, y*
- *por otro lado los datos de cada visita que la mascota hace a la clínica y del tratamiento que se le indica en cada caso.*

Con la estructura que ***Víctor*** *ha hecho, cada vez que una mascota hace una nueva visita a la clínica, tendríamos que añadir una nueva tupla a la tabla correspondiente, en la que habría que meter además de los datos de la visita* **`(fecha_visita, enfermedad, peso, tratamiento)`** *todos los demás datos de la mascota* **`(cod_mascota, nombre, raza, dirección, telefono, color)`**. *Por tanto, si esta mascota ha venido 10 veces, ¡todos esos datos estarían repetidos 10 veces!*

A ***Víctor*** *le cuesta reconocer que su diseño estaba mal, y alega que tampoco es tan grave tener que almacenar 10 veces los datos de la mascota, ya que hoy en día cualquier ordenador puede tener una enorme capacidad de almacenamiento en su disco duro, por precios que casi son de risa.*

José *y* ***María*** *le dicen que no están del todo de acuerdo. Si en vez de hablar de una clínica veterinaria con cientos de clientes como mucho estuviéramos hablando de la base de datos clínicos del SAS (Servicio Andaluz de Salud) en la que hay millones de usuarios, que hacen probablemente cientos o incluso miles de visitas médicas a lo largo de su vida, tener que almacenar duplicada la información personal de cada usuario tendría un coste nada despreciable. Esto convence bastante a* ***Víctor****, pero el desperdicio de espacio de almacenamiento no es sin embargo el problema principal, le comenta* ***José****.*

Volviendo al caso de la clínica veterinaria, ¿Qué pasaría si una mascota cambia por ejemplo de dirección? Evidentemente, para que la información de la base de datos fuera correcta, habría

que actualizar la dirección en todas las tuplas de la relación, lo que sin duda haría que la aplicación invirtiera mucho tiempo actualizando un conjunto grande de tuplas, en lugar de actualizar una sola, lo que incidiría en el rendimiento general de la aplicación. De nuevo podemos pensar en el ejemplo del SAS para ver lo grave que el problema puede ser. Piensa en miles de médicos esperando a que la aplicación les responda para poder hacer su trabajo, porque está saturada haciendo miles de actualizaciones realmente redundantes en cientos de miles de registros de los usuarios.

*A estas alturas **Víctor** ya está totalmente convencido del error, pero **José** le dice que esto último, aún siendo grave, no es tampoco el problema principal. Imagina que al actualizar las tuplas de la relación, no se ha hecho bien. Existe la posibilidad de que la misma mascota aparezca con una dirección distinta en tuplas distintas, lo que sería una inconsistencia. No tendríamos forma de saber cuál es la dirección correcta, por lo que ninguna nos resultaría fiable. Estas **inconsistencias**, derivadas de la redundancia de la información, serían gravísimas, porque en la práctica supondrían que la información de la base de datos está corrupta.*

Vale, vale, protesta Víctor.

Ya está más que convencido de que hay que hacer un buen diseño, y que para ello es necesario evitar esos problemas, pero ¿cómo se consigue?

- *Aplicando la teoría de la normalización, que es algo que **María**, con la ayuda de **Carmen**, se van a encargar de enseñarte, le dice José. Porque debes saber que todavía hay más problemas que los mencionados derivados de un mal diseño, como imposibilidad de representar ciertos datos del problema, violación de la integridad de la base de datos en los borrados, etc.*
- *Y por curiosidad, ¿Cómo quedaría el diseño de la clínica para evitar esos problemas?, pregunta **Víctor**.*
- *Sencillo. Basta con dividir la relación en dos nuevas relaciones, de forma que hechos distintos se almacenen en objetos distintos, aplicando, claro está, la teoría de la normalización para que la división sea correcta:*

```
MASCOTA (cod_mascota, nombre, direccion, telefono, color, raza)

VISITA (cod_mascota, fecha_visita, peso, enfermedad, tratamiento)
```

Como hemos visto en unidades anteriores, uno de los principales objetivos del diseño de la base de datos es conseguir una **estructura lógica** que:

- Evite problemas en inserciones, modificaciones y borrados, por lo tanto, no sufra anomalías de almacenamiento.
- El modelo lógico y físico puedan modificarse con facilidad si hay una variación de los requerimientos iniciales, y además que ambos modelos sean independientes el uno del otro.

No es muy difícil intuir que, en todos los modelos teóricos, el diseño de una base de datos se puede realizar de dos maneras diferentes:

- Obteniendo el **esquema relacional** directamente a partir de la información que poseemos acerca del problema a modelizar, de tal manera que obtendremos un conjunto de esquemas de relación que contendrán los atributos y las restricciones de integridad que representan el análisis que hemos realizado de nuestro problema a tratar.
- Realizando el diseño en **dos fases**, la primera llevando a cabo el diseño conceptual, es decir, obteniendo el diagrama E/R, y en la segunda fase, transformando éste en un esquema

relacional como hemos visto en la unidad anterior.

Siguiendo cualquiera de los dos métodos el resultado final es un esquema relacional que debe verificar en todo momento un principio básico en todo diseño:

> ***Hechos distintos se deben almacenar en objetos distintos.***

Es evidente que la siguiente relación no cumple con este principio básico:

MASCOTA (cod_mascota, nombre, raza, direccion, telefono, enfermedad, color, peso, fecha_visita, tratamiento)

En principio los atributos enfermedad, fecha_visita y tratamiento no explican el mismo "hecho" que el resto, ya que tratan de las visitas, enfermedades y tratamiento de la mascota, mientras que el resto de los atributos da información acerca de los datos concretos de la mascota.

Además podemos observar que con este diseño no podríamos insertar una nueva mascota en la base de datos de la clínica veterinaria hasta que no hiciera su primera visita.

En lugar de la relación anterior podríamos haber diseñado el siguiente esquema relacional:

MASCOTA (cod_mascota, nombre, direccion, telefono, color, raza)

VISITA (cod_mascota, fecha_visita, peso, enfermedad, tratamiento)

Éste sí verifica el principio básico del diseño de Bases de Datos, y permite insertar mascotas nuevas en la base de datos independientemente de que realice una visita a la clínica veterinaria.

Para evitar dichas anomalías y conseguir los objetivos mencionados anteriormente se aplica la Teoría de la Normalización.

Nos preguntamos entonces, ¿qué es **la Teoría de la Normalización**?

La Teoría de la Normalización (o simplemente la normalización) consiste en aplicar una serie de **reglas** para asegurarnos la eliminación de redundancias e inconsistencias en las tablas que constituyen el diseño de nuestra base de datos. En ocasiones esta Normalización se traduce en la separación de los datos en diferentes tablas asegurándonos siempre de que las tablas resultantes mantengan toda la información original y las distintas dependencias entre los datos almacenados.

Todo esto se traduce en la aplicación de las Formas Normales**, que sólo son un conjunto de restricciones sobre las tablas que evitan los problemas de redundancia y anomalías de modificación, inserción y borrado de datos.**

De esta manera transformamos, si es necesario, las tablas obtenidas en el Modelo Relacional en un conjunto de tablas más simples y fáciles de mantener, tanto desde el punto de vista lógico, como físico.

Seguramente con el tiempo y la práctica conseguirás obtener el esquema relacional directamente del enunciado del problema, pero aunque esto sea así, es recomendable comprobar el diseño obtenido con la teoría de la normalización.

Resumiendo lo visto hasta el momento, **la normalización consiste en una descomposición de la relación universal** (toda la información en una única tabla) **o de una colección de relaciones equivalentes a la misma, en una colección de relaciones en la que las anomalías de**

actualización (inserción, borrado y modificación) no existan o sean mínimas.

Las reglas en las que se basa la Teoría de la Normalización son conocidas con el nombre de **Formas Normales**.

Existen **seis formas normales**. Las tres primeras reglas de normalización fueron perfiladas por el Dr. E.F.Codd en su escrito de 1972, "Further Normalization of the Data Base Relational Model" (Referente a la normalización de las Bases de Datos Relacionales).

Posteriormente, y como consecuencia de unas anomalías detectadas al utilizar las tres primeras formas normales, Boyce ayudó a Codd a redefinir la tercera forma normal en lo que se conoce como la forma normal de Boyce-Codd. Más tarde Ronald Fagin introdujo la cuarta y quinta forma normal en un intento de mejorar el rendimiento de grandes sistemas transaccionales de las bases de datos modernas.

Para saber más

Si quieres saber algo más acerca del precursor de la teoría de Normalización, pulsa el siguiente enlace donde encontrarás más cosas acerca de sus aportaciones al mundo de la informática:

Ronald Fagin

__María__ y __Carmen__ ya se han puesto a la tarea, y han comenzado a instruir a __Víctor__ en la teoría de la normalización de bases de datos. El primer punto es detallarle los __objetivos__ que se persiguen con la normalización y las ventajas que presenta su uso.

Tras la charla mantenida con __José, Víctor__ cree tenerlo bastante claro, pero de todas formas, se encuentra con alguna cuestión nueva, que le resulta muy interesante para tener en cuenta, como por ejemplo, el hecho de que al normalizar se debe cuidar que se conserve toda la información del problema, así como las dependencias existentes entre distintos datos, sin que se introduzcan dependencias nuevas que no existían originalmente en el problema real.

Pero lo que más le ha llamado la atención es que no siempre es posible conseguir todos esos objetivos a la vez. Por tanto, la normalización será un camino para conseguir alcanzar, de forma sistemática, el mayor número de ellos.

Las principales ventajas del uso de la normalización son las siguientes:

- **Facilidad de uso y claridad**, las tablas recogen los datos de manera clara y sencilla de entender incluso para usuarios poco expertos.
- **Flexibilidad y facilidad de gestión**, la información que necesitan los usuarios se puede obtener directamente de las tablas o mediante operaciones de álgebra y cálculo relacional.
- **Precisión**, las relaciones entre las tablas hacen que los datos de distintas tablas se relacionen con toda exactitud.
- **Mínima redundancia**, no se guarda información duplicada de manera innecesaria.
- **Máximo rendimiento de las aplicaciones**, sólo se utiliza la información que va a servir de utilidad a cada aplicación.

Los principales **objetivos** que busca un diseño normalizado son los siguientes:

- Eliminar anomalías de actualización, inserción y borrado.

- Conservar la información original (descomposición sin pérdida de información).
- Conservar las dependencias funcionales originales (descomposición sin pérdida de dependencias funcionales).
- No crear dependencias nuevas o relaciones inexistentes.
- Facilidad de uso y modificación de tablas creadas.
- Eficiencia.

Pero siempre tenemos que tener en cuenta una cosa:

¡¡A veces no es posible conseguir todos esos objetivos a la vez!!

***María** le explica a **Víctor** que gran parte del proceso de normalización se basa en el concepto de **dependencia funcional**, por lo que es muy importante que éste le quede totalmente claro, y se lo explica con detalle. Pero no existe un único tipo de dependencia, por lo que es necesario instruirle en los conceptos de dependencia plena o completa, trivial, elemental, transitiva y transitiva estricta. Detectar las dependencias existentes entre los datos que forman parte de nuestro problema es fundamental, ya que son la base sobre la que aplicar las distintas técnicas para conseguir que todas las tablas de nuestra base de datos estén en una determinada forma normal que nos hemos propuesto como objetivo. Para que lo entienda mejor, **Carmen** le propone un ejemplo en el que se trabaja con empleados asignados a proyectos, de forma que se van identificando las distintas dependencias existentes en cada caso.*

Todo lo que hemos comentado en el apartado anterior está muy bien como introducción, pero ¿en qué se basa toda esta Teoría de la Normalización?

Pues en un concepto muy sencillo y que ha permitido un gran avance en el diseño de las bases de datos: el concepto de **dependencia funcional.**

Sabemos que una relación está formada por un conjunto de atributos, pero ahora afinamos un poco más y completamos la definición:

Una relación se compone de un conjunto de atributos y dependencias, R (A, DF). Los atributos (A) sabemos cómo identificarlos, pero las dependencias (DF), ¿cómo sabemos cuáles son las dependencias y cómo las identificamos?

Las **dependencias funcionales** son propiedades de la semántica de los atributos y no pueden obtenerse de manera automática en una relación determinada.

Definición de dependencia

Las dependencias funcionales son:

- **Restricciones de integridad** establecidas por el usuario que permiten conocer las relaciones que existen entre los atributos del problema planteado.
- **Propiedades inherentes** a la semántica del problema.
- **Se han de cumplir para todos los registros de una relación**.

Resumiendo, las **dependencias funcionales reflejan enlaces semánticos permanentes entre los datos en un diseño concreto**.

En el siguiente apartado, al definir los tipos de dependencias, te presentamos algunos ejemplos que aclaran este concepto.

Tipos de dependencias (I)

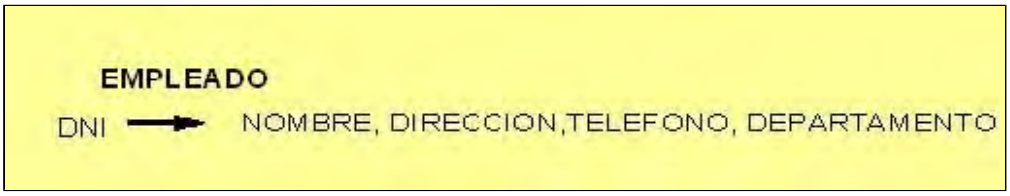
Consideramos un esquema de relación general **R(A)**, con A el conjunto de sus atributos, y consideramos X, Y, y Z subconjuntos de A llamados descriptores. Nos encontramos con que podemos definir los siguientes tipos de dependencias:

- **Dependencia funcional.**

Se dice que un conjunto de atributos (**Y**) **depende funcionalmente** de otro conjunto de atributos (**X**), si para cada valor de **X** existe un único valor posible para **Y**. Se nota por **X(Y**. También se dice que **X determina a Yo** que **X implica a Y**. A **X** se le conoce como determinante o implicante y a **Y** se le conoce como implicado.

Por ejemplo, si consideramos la relación **EMPLEADO (DNI, NOMBRE, DIRECCION, TELEFONO, DEPARTAMENTO)** es evidente que el nombre de una persona depende funcionalmente del **DNI**, es decir, para un **DNI** determinado existe sólo un nombre que le corresponda. Sin embargo, si consideramos el ejemplo al revés, **DNI** no depende funcionalmente de **NOMBRE**, ya que para un mismo nombre puede haber muchos **DNI** diferentes. ¡Piensa en la cantidad de "Pepe y José" que hay en este país... y cada uno tiene su DNI!

En este caso, suponemos que cada persona tiene un único teléfono y una única dirección, y que además trabaja en un único departamento de la empresa. Por tanto, las dependencias funcionales quedan como siguen:



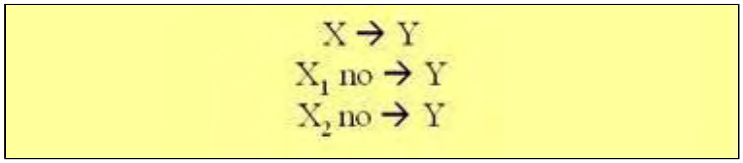
No obstante, si en nuestro problema necesitáramos **almacenar** (o quisiéramos permitir que se almacenara) más de una dirección para una misma persona, o más de un teléfono, o que trabajara en más de un departamento, esos tres atributos no dependerían funcionalmente de **DNI**. A eso es a lo que nos referimos cuando decimos que las dependencias funcionales vienen definidas por la semántica del problema.

- **Dependencia funcional plena o completa.**

Si el descriptor **X** es compuesto, **X(X_1, X_2),** se dice que **Y tiene dependencia funcional completa o plena de X**, si depende funcionalmente de **X**, pero no depende funcionalmente de ningún subconjunto de **X**, es decir:

Y depende funcionalmente de **X**, pero **Y** no depende funcionalmente de X_1, ni tampoco de X_2.

Se representa:



Una dependencia funcional completa se denota como X Y

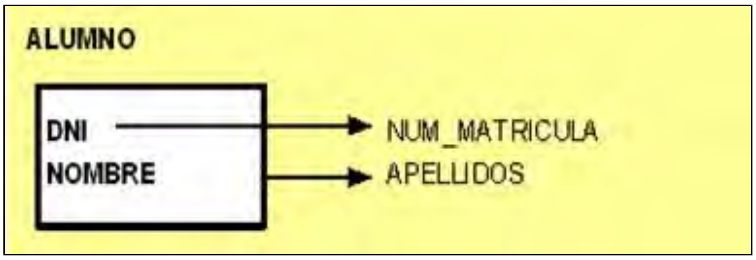
Si consideramos la relación:

PROYECTO (COD_PROYECTO, DNI_EMPLEADO, FECHA_INCORPORACION, HORAS_DEDICADAS)

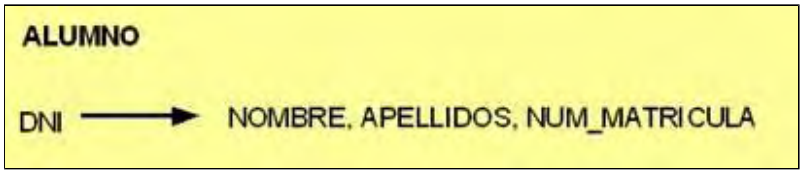
está claro que los atributos **FECHA_INCORPORACION** y **HORAS_DEDICADAS** tienen dependencia funcional completa respecto de **COD_PROYECTO** y **DNI_EMPLEADO**. Es evidente que la **FECHA_INCORPORACION** a un proyecto de un empleado depende tanto del proyecto al que se incorpora como del empleado que se incorpora. Un mismo empleado puede pertenecer a varios proyectos a los que se incorpora en diferentes fechas. Igual ocurre con el atributo **HORAS_DEDICADAS**.

La dependencia funcional completa se puede representar mediante un diagrama de dependencias funcionales como sigue:

En cambio, si consideramos la relación **ALUMNO (DNI, NOMBRE, APELLIDOS, NUM_MATRICULA)**, el conjunto de atributos formado por el **NOMBRE** y el **DNI** producen dependencia funcional sobre el atributo **APELLIDOS**, y si consideramos que el **NUM_MATRICULA** depende únicamente del atributo **DNI** podemos representarlo como sigue:



Pero la dependencia de los atributos **DNI** y **NOMBRE** no es completa ya que **DNI** sólo, también produce dependencia funcional sobre **APELLIDOS**. De esta manera sólo el atributo **DNI** produce una dependencia funcional completa sobre el campo **APELLIDOS**. Es más, **DNI** produce una dependencia funcional completa sobre el resto de los atributos. Por lo que la dependencia real de la relación **ALUMNO** queda:



Tipos de dependencias (II)

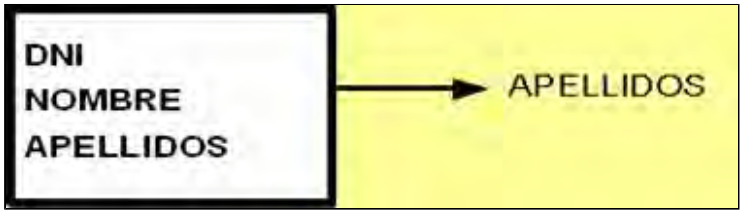
¿Pero hay más tipo de dependencias funcionales?

Por supuesto. Seguimos mostrándotelas.

- **Dependencia funcional trivial.**

Una dependencia funcional X --> Y se dice que es trivial si Y es un subconjunto de X (Y c X)

Si consideramos el conjunto de atributos **NOMBRE, APELLIDOS, DNI** y consideramos la dependencia funcional



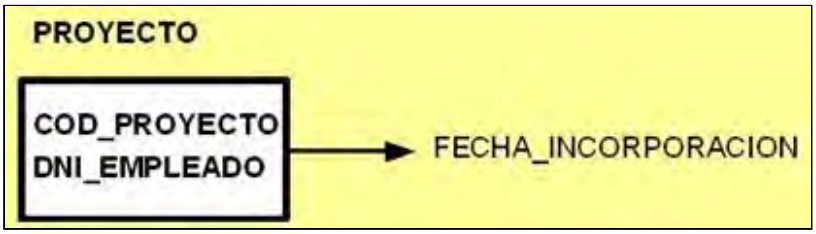
dicha dependencia es trivial.

No debe haber dependencias funcionales triviales en las relaciones de las bases de datos.

- **Dependencia funcional elemental.**

Una dependencia funcional **X-->Y** es **elemental** si **Y** es un atributo único no incluido en **X**, y no existe ningún subconjunto de **X**, llamémosle X_1, tal que X_1 **(** **Y**, es decir, **la dependencia funcional elemental es una dependencia completa no trivial en la que el implicado es un atributo único**.

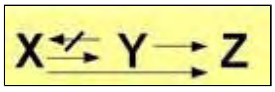
Si consideramos la relación **PROYECTO (COD_PROYECTO, DNI_EMPLEADO, FECHA_INCORPORACION)** con **FECHA_INCORPORACION** dependiendo funcionalmente de **COD_PROYECTO** y **DNI_EMPLEADO**, vemos claramente que esta dependencia es una **dependencia funcional elemental** ya que el implicado está constituido únicamente por el atributo **FECHA_INCORPORACION**.



- **Dependencia funcional transitiva.**

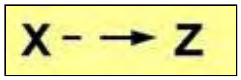
Consideramos la relación **R(X, Y, Z)** con las siguientes dependencias funcionales:

Es decir:



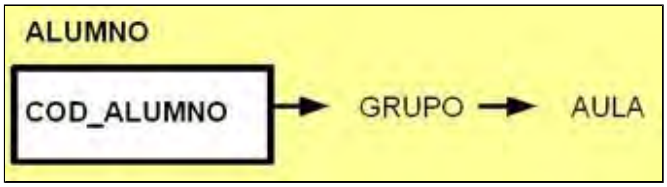
- **Z** depende funcionalmente de **X** y de **Y**,
- **Y** depende funcionalmente de X, pero
- **X** no depende funcionalmente de **Y**.

Se nota de la siguiente manera:



En estas condiciones se dice que **Z** tiene una **dependencia funcional transitiva** respecto a **X**, a través de **Y**. Es decir, un descriptor **Z** es transitivamente dependiente de otro **X** si se puede conoce por diferentes vías, una directamente y otra a partir de otro descriptor intermedio **Y**.

Si consideramos la siguiente relación **`ALUMNO (COD_ALUMNO, GRUPO, AULA)`** claramente se aprecia que los atributos **`GRUPO`** y **`AULA`** dependen funcionalmente del atributo **`COD_ALUMNO`**, y a su vez el atributo **`AULA`** depende funcionalmente del atributo **`GRUPO`**, por lo que **`AULA`** tiene una dependencia funcional transitiva respecto a **`COD_ALUMNO`** a través de **`GRUPO`**.

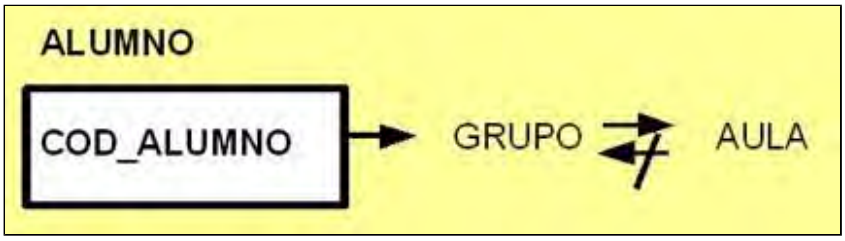


- **Dependencia funcional transitiva estricta.**

Una dependencia funcional transitiva se dice que es estricta si se verifica que

$$Y \not\rightarrow Z$$

En el caso anterior el atributo **`GRUPO`** no depende funcionalmente de **`AULA`**.



- **Dependencia funcional multivaluada.**

Dada una relación **R (X, Y, Z)** con los atributos **X**, **Y** y **Z**, la **dependencia multivaluada R.X -->--> R.Y** se cumple en **R** si y sólo si el conjunto de valores de **Y** correspondiente a un par dado (valor de **X**, valor de **Z**) en **R** depende sólo del valor de **X** y es independiente del valor de **Z**.

Si consideramos la relación **`AGENDA (DNI, DIRECCION, FECHA_NACIMIENTO, TELEFONO)`** podemos observar que por cada **`DNI`** de la **`AGENDA`** puedo tener varios números de **`TELEFONO`** independientemente de los valores que tomen **`dirección`** y **`FECHA_NACIMIENTO`**. Podemos decir que **`TELEFONO`** es un atributo multivaluado y que existe la siguiente dependencia multivaluada **`DNI -->--> TELEFONO`**

DNI	DIRECCION	FECHA_NACIMIENTO	TELEFONO
24222425	Islas Vírgenes, 20	19-04-1975	950202120
24222425	Islas Vírgenes, 20	19-04-1975	67676869
24222425	Islas Vírgenes, 20	19-04-1975	950343536
71734746	Granada, 18	21-06-1969	958192087
71734746	Granada, 18	21-06-1969	667667667
71734746	Granada, 18	21-06-1969	958950950

Para saber más

En esta unidad únicamente se da una definición general del concepto de dependencia funcional multivaluada, pero si quieres profundizar más, en los siguientes enlaces encontrarás más información:

Dependencia funcional multivaluada Dependencia funcional multivaluada **[Versión en caché]**

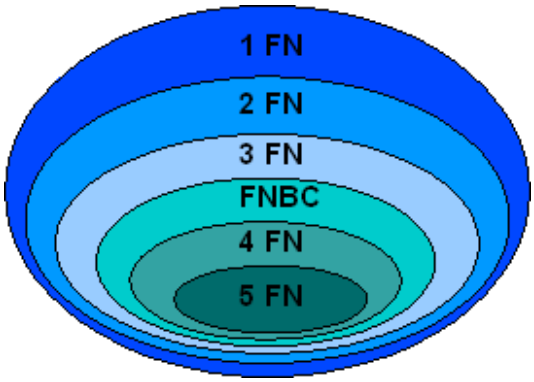
Una vez entendido el concepto de dependencia funcional, y los distintos tipos de dependencias que existen, ha llegado la hora de aplicarlo para conseguir normalizar nuestra base de datos.

*En este punto, **María** le explica a **Víctor** que existen distintos "grados de normalización", que se van diferenciando entre sí, fundamentalmente por el grado de redundancia de la información que permiten. Mientras más alto sea el grado de normalización, menos redundancia de información habrá, y menos anomalías o problemas a la hora de insertar, actualizar o borrar información en la base de datos, reduciéndose las posibilidades de errores o inconsistencias.*

*Esos "grados de normalización" se conocen como **Formas Normales**, y nuestro objetivo es ir asegurándonos de que todas las tablas de nuestro esquema van cumpliendo cada una de esas formas normales, y si no es así, aplicar los criterios de normalización para introducir las modificaciones necesarias en el esquema para que sí lo estén.*

***Carmen** le comenta a **Víctor** que no siempre es posible conseguir el máximo grado de normalización. Siempre es posible llegar a la Tercera Forma Normal, y por tanto es el mínimo exigible. Sin embargo, no siempre es posible conseguir las siguientes formas normales, aunque debemos intentarlo. Por **ejemplo**, al descomponer la relación para que esté en FNBC, a veces se pierden dependencias funcionales, por lo que no sería recomendable pasar de 3FN. Usualmente, basta con conseguir la 3FN o la Forma Normal de Boyce-Cood. Raramente se normaliza hasta la 4FN, y aunque existe la 5FN, es posible conseguirla en un número reducido de casos muy específicos, que habrá que analizar cuidadosamente, y que rara vez se presentan, por lo que no merece la pena, a priori, considerarlos para nuestras aplicaciones de gestión típicas.*

*Para practicar, **Carmen** le propone a **Víctor** un ejercicio de normalización para una base de datos de alumnos, asignaturas y calificaciones. **Víctor** tendrá que ir comprobando si está en 1FN, luego conseguir que esté en 2FN, hacer los cambios oportunos para que esté en 3FN, e intentar conseguir la FNBC.*



Hasta ahora hemos definido los **conceptos** necesarios para poder entender las distintas formas normales, pero de aquí en adelante vamos a conocer dichas formas normales y los mecanismos para transformar las relaciones en otras equivalentes normalizadas si las dadas no lo estuvieran.

Según vayamos conociendo las distintas formas normales podremos comprobar que todas ellas van dependiendo de las anteriores, tal y como se refleja en la imagen con la que comienza el apartado.

Como vemos en esa imagen del comienzo del apartado, existen más formas normales aparte de las que veremos en este apartado, y todas ellas continúan verificando las dependencias que se indican en la figura.

La **normalización** a partir de la FNBC rara vez se produce, ya que sólo hace falta en sistemas muy

concretos e inusuales. En algunos libros se habla hasta de una 6, 7 y 8FN, pero dichas formas normales se salen de los objetivos de esta unidad, y de este módulo profesional.

1FN (Primera Forma Normal)

> ***Una relación está en primera forma normal (1FN) si y sólo si todos sus atributos son atómicos.***

La siguiente relación no estaría en primera forma normal:

DNI	DIRECCION	TELEFONO
76757473	Granada, 21	950343536 667667667
24252627	Islas Vírgenes, 20	950505050 950343434 696789210

El atributo **`TELEFONO`** no es **atómico**, ya que para cada algunas tuplas tiene más de un valor. ¿Cómo podríamos solucionar el problema? De manera intuitiva surgen dos posibles soluciones:

- Si sabemos el número exacto de distintos teléfonos que puede tener cada registro de la tabla, podemos añadir tantos atributos como teléfonos se necesiten.

DNI	DIRECCION	TELEFONO_TRABAJO	TELEFONO_MÓVIL	TELEFONO_CASA
76757473	Granada, 21	950343536	667667667	
24252627	Islas Vírgenes, 20	950343434	696789210	950505050

  El problema de esta posible solución es que no es óptima prque desaprovecha mucho espacio de almacenamiento. Imaginemos por ejemplo a varias personas que sólo tienen un teléfono, por ejemplo, el teléfono del trabajo, pero no tienen ni teléfono móvil, ni teléfono de casa. Por cada una de estas personas estaremos desaprovechando un espacio reservado para almacenar el teléfono móvil y el teléfono de casa.

- Creando una nueva tabla, en la cual almacenemos cada uno de los teléfonos que tenga la persona, de la siguiente manera: conservamos la relación original prescindiendo del campo multivaluado, y creamos una nueva relación que estará compuesta por la clave primaria (el dni en este caso) y el campo que contenía múltiples valores (en este caso el teléfono). De este modo, en esta nueva relación, la clave primaria estará formada por la unión de ambos atributos, es decir, por la pareja dni y teléfono. Podemos apreciar también, que lo que si antes se almacenaban tres valores en una celda, ahora eso se traduce en tener tres filas en la nueva relación.

DNI	DIRECCION
76757473	Granada, 21

24252627	Islas Vírgenes, 20

DNI	TELEFONO
76757473	950343536
76757473	667667667
24252627	950505050
24252627	950343434
24252627	696789210

En ambos casos las tablas resultantes sí están en primera forma normal, aunque reiteramos, que la mejor forma de normalizar sería la segunda explicada.

2FN (Segunda Forma Normal)

> ***Una relación está en segunda forma normal (2FN) si y sólo si está en 1FN y todos los atributos no clave dependen por completo de la clave primaria.***

Consideramos la siguiente relación:

ALUMNO (DNI, COD_ASIGNATURA, NOMBRE, APELLIDOS, NOTA, CURSO, AULA)

donde se aprecian las siguientes dependencias funcionales:

DNI --> NOMBRE, APELLIDOS

DNI, COD_ASIGNATURA --> NOTA

COD_ASIGNATURA --> CURSO, AULA

Dicha relación **NO se encuentra en 2FN** al estar los atributos **NOMBRE** y **APELLIDOS** determinados sólo por el atributo **DNI** y no por la totalidad de la clave principal. Igual ocurre con los atributos **CURSO** y **AULA**, que están determinados sólo por **COD_ASIGNATURA**. Este hecho presenta una serie de inconvenientes tales como:

- **Redundancia de datos**: Con cada alumno de una misma asignatura repetimos toda la información acerca de la asignatura.
- **Anomalías en la inserción de tuplas**: Tenemos anomalías de inserción ya que no se pueden insertar las asignaturas que se imparten en un curso hasta que no insertamos algún alumno matriculado en esas asignaturas. En caso contrario, estaríamos violando la regla de integridad de la clave al existir valores nulos en dicha clave.
- **Anomalías en el borrado de tuplas:** Igualmente aparecen anomalías en el borrado de tuplas, ya que al eliminar todos los alumnos de una determinada asignatura queda eliminada de manera automática dicha asignatura.
- **Anomalías en la actualización de tuplas:** Si se hace un cambio de aula para una asignatura, es necesario modificar todas las tuplas de todos los alumnos matriculados en esa asignatura. Volvemos a ver que existe una gran redundancia de información en la Base de Datos.

- **Posible inconsistencia de datos en las actualizaciones:** No es más que una consecuencia de las anomalías en la actualización de tuplas. Imagina que se te pasa modificar todas las tuplas de todos los alumnos matriculados en una asignatura cuando se producen cambios en la asignatura (por ejemplo, cambia el aula donde se imparte la asignatura). Alumnos distintos

podrían tener datos distintos para una misma asignatura, por ejemplo, distintas aulas donde se imparte, cuando realmente se imparte en una única aula. Eso sería una inconsistencia.

- **Imposibilidad de almacenar ciertos datos.** No es más que una consecuencia de las anomalías en la inserción. No nos resultaría posible representar datos sobre asignaturas en las que no haya matriculado ningún alumno, o sobre aulas en las que no se imparte ninguna asignatura, cuando es perfectamente posible que queramos almacenar esa información.

Pero ¿qué se hace cuando una relación **R** no está en 2FN?

Se transforma para que sí esté en 2FN, y esta transformación se realiza de la siguiente manera:

> ***Dada una Relación R con atributos X, Y, Z simples o compuestos, con la clave (X, Y) si en la relación existe una dependencia funcional incompleta Y --> Z (Z depende de parte de la clave), entonces:***
>
> - ***De R se elimina Z.***
> - ***Se construye una nueva relación R' con:***
>   - ***Z como atributo no clave.***
>   - ***Y como clave primaria de R'.***

Las relaciones **R** y **R'** resultantes sí están ahora en segunda forma normal (2FN).

Veamos esto con más claridad con ayuda de un ejemplo. Consideramos la siguiente relación y comprobamos si está en 2FN:

ALUMNO (DNI, COD_ASIGNATURA, NOMBRE, APELLIDOS, NOTA, CURSO, AULA)

Lo primero que tenemos que comprobar es que dicha relación está en primera forma normal, como sólo tenemos la intención de la relación, damos por supuesto que todos los valores de los atributos son atómicos. Ahora hay que comprobar la segunda parte de la definición de segunda forma normal, es decir, que todos los atributos no clave dependen por completo de la clave primaria.

Es decir, se tienen que verificar las siguientes dependencias:

DNI, COD_ASIGNATURA --> NOMBRE

DNI, COD_ASIGNATURA --> APELLIDOS

DNI, COD_ASIGNATURA --> NOTA

DNI, COD_ASIGNATURA --> CURSO

DNI, COD_ASIGNATURA --> AULA

Pero es evidente que los atributos **NOMBRE** y **APELLIDOS** no dependen por completo de la clave, ya que sólo dependen de **DNI**, y no de **COD_ASIGNATURA**.

¿Cómo solucionamos esta situación?

Sencillamente **descomponiendo** la relación dada en otras relaciones que sí estén en segunda forma normal, separando los atributos que no tienen dependencia funcional completa respecto de la clave principal en otra tabla, junto con el/los atributo/s que forman la clave principal con los que sí tienen dependencia funcional completa.

En nuestro caso concreto, la descomposición sería la siguiente:

ALUMNO (DNI, NOMBRE, APELLIDOS)

CALIFICACION (DNI, COD_ASIGNATURA, NOTA)

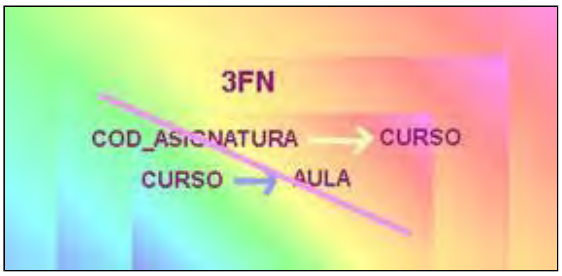
ASIGNATURA (COD_ASIGNATURA, CURSO, AULA)

Como podemos comprobar, ahora **NOMBRE** y **APELLIDOS** sí tienen dependencia funcional completa respecto de **DNI** en la relación **ALUMNO**, y en el caso de la relación **CALIFICACION**, el atributo **NOTA** depende completamente de la clave principal compuesta por **DNI** y **COD_ASIGNATURA**. Igualmente ocurre en el caso de la relación **ASIGNATURA**, en la que tanto **CURSO** como **AULA** tienen dependencia funcional completa respecto de **COD_ASIGNATURA**.

Por lo tanto, las tres relaciones están en 2FN, habiendo conseguido **conservar las dependencias funcionales originales, la información y solucionado todos los problemas mencionados anteriormente**.

FN (Tercera Forma Normal)

Una relación está en tercera forma normal (3FN) si y sólo si esta en 2FN y todos los atributos no clave dependen de manera no transitiva de la clave primaria.



Es decir, si sus atributos no clave son **mutuamente independientes** (no existe ningún atributo no clave que dependa funcionalmente de alguna combinación del resto de atributos no clave) y además sean por completo dependientes funcionalmente de la clave primaria.

Consideramos la relación ASIGNATURA del ejemplo anterior:

ASIGNATURA (COD_ASIGNATURA, CURSO, AULA)

En la que existen las siguientes dependencias funcionales:

COD_ASIGNATURA --> CURSO

CURSO --> AULA

Con estas dependencias surgen una serie de inconvenientes en la relación:

- **Anomalías en la inserción de tuplas:** no se puede insertar un curso si no se indica al menos una de las asignaturas de dicho curso. Estaríamos violando la regla de integridad de la clave a insertar una tupla con valores nulos en la clave primaria.
- **Anomalías en el borrado de tuplas:** si se borran las tuplas de todas las asignaturas de un curso concreto, se borra de manera automática dicho curso de la Base de Datos.
- **Anomalías en la actualización de tuplas:** si se realiza un cambio de aula para un curso determinado, tenemos que cambiar todas las tuplas de las asignaturas de dicho curso, lo que indica que existe una gran redundancia de información en la Base de Datos.

De esta manera volvemos a cuestionarnos cómo solucionar que una relación R no esté en 3FN... Al igual que en el caso de la 2FN, la solución pasa por una descomposición de la relación dada en otras relaciones que sí estén en 3FN. Esta descomposición se realiza de la siguiente manera:

Dada una relación R con atributos X, Y, Z simples o compuestos, con la clave X y atributos no claves Y, Z tal que en la relación existen las dependencias funcionales:

X --> Y

Y <-- Z

... es decir, una dependencia funcional transitiva X --> Z, entonces la relación R se descompone como sigue:

- ***De R se elimina Z (el atributo que tiene la dependencia funcional transitiva en la relación).***
- ***Se construye una nueva relación R'con:***
  - ***Z como atributo, pero que no sea clave principal.***
  - ***Y como clave principal de R' .***
  - ***Y --> Z como dependencia funcional.***

Las relaciones **R** y **R'** resultantes sí están en tercera forma normal. En definitiva, hemos eliminado las dependencias transitivas.

Veamos esta descomposición con el ejemplo anterior:

ASIGNATURA (COD_ASIGNATURA, CURSO, AULA)

Lo primero que tenemos que hacer es comprobar si la relación está en 2FN, para lo que se tiene verificar que está en 1FN y que los atributos no clave tienen dependencia funcional completa respecto de la clave principal.

Damos por cierto que todos los atributos toman valores atómicos para que la relación esté en 1FN.

Comprobamos ahora que en efecto, tanto **CURSO** como **AULA** tienen dependencia funcional completa respecto de **COD_ASIGNATURA**, por lo que la relación **ASIGNATURA** está en 2FN.

Sólo nos queda comprobar si existen dependencias funcionales transitivas, en cuyo caso la relación dada no estaría en 3FN.

Como podemos ver existen las siguientes dependencias en la relación:

COD_ASIGNATURA --> CURSO

CURSO --> AULA

... por lo que existe una **dependencia funcional transitiva** en la relación.

¿Cómo eliminamos la dependencia funcional transitiva?

Descomponiendo la relación dada en otras de tal manera que desaparezca dicha dependencia, es decir, separando el/los atributo/s que producen la dependencia funcional transitiva en una nueva relación de la siguiente manera:

ASIGNATURA (COD_ASIGNATURA, CURSO)

CURSO (CURSO, AULA)

FNBC (Forma Normal de Boyce-Codd)

> ***Una relación está en forma normal Boyce/Codd (BCNF) si y sólo si todo determinante es una clave candidata.***

Si consideramos la siguiente relación:

CALLEJERO (CALLE, CIUDAD, COD_POSTAL)

con las siguientes dependencias funcionales:

CALLE, CIUDAD --> COD_POSTAL

COD_POSTAL --> CIUDAD

COD_POSTAL es un determinante que no es clave candidata, por lo que la relación no está en **FNBC**.

En esta ocasión vuelven a aparecer **anomalías** de inserción, actualización y borrado en la Base de Datos, y dichas anomalías desaparecen realizando una descomposición adecuada de la relación. En ocasiones esta descomposición puede dar lugar a una pérdida de dependencias funcionales, motivo por el que algunos autores no aconsejan pasar a la FNBC y quedarse en la 3FN. Otros sin embargo, prefieren continuar el proceso de normalización hasta sus formas más avanzadas.

En caso de decidir normalizar hasta la FNBC, la descomposición se realiza de forma genérica como sigue:

> ***Dada una relación R con atributos X, Y, Z simples o compuestos, con clave {X, Y} y tal que en la relación existen las siguientes dependencias funcionales:***

{X, Y} --> Z

Z --> Y

es decir, existe un determinante no clave, y por lo tanto la relación no está en forma normal de Boyce/Codd. Dicha relación R se descompone como sigue:

- ***En R se deja la parte de la clave que es independiente y todos los atributos no primarios, es decir, R (X, Z)***
- ***Se crea otra tabla R' con la parte de la clave restante y el atributo secundario del que depende, siendo éste último la clave de la nueva tabla. Es decir, R' (Y, Z).***

Las relaciones **R** y **R'** resultantes sí están en forma normal de Boyce/Codd.

Vemos esta descomposición con la relación **CALLEJERO**.

¿Cómo podemos hacer que dicha relación esté en forma normal Boyce /Codd?

Muy sencillo, siguiendo los pasos indicados más arriba. Tenemos que separar los atributos origen del conflicto, para ello creamos las relaciones:

- En **R** dejamos la parte de la clave de **CALLEJERO** que es independiente, es decir, **CALLE**, y todos los atributos no primarios. En este caso sólo tenemos el atributo **COD_POSTAL**. Por lo que **R** queda:
  CALLEJERO' (CALLE, COD_POSTAL)
- En **R'(CODIGO_POSTAL)** consideramos el resto de la clave primaria de la relación **CALLEJERO**, es decir, **CIUDAD**, y el atributo del que depende, en este caso **COD_POSTAL**, con lo que **R' (CODIGO_POSTAL)** nos queda:
  CODIGO_POSTAL (COD_POSTAL, CIUDAD)

Ahora **CALLEJERO'** y **CODIGO_POSTAL** sí están en forma normal de Boyce/Codd.

No es objeto de esta unidad profundizar en los conceptos relacionados con la cuarta y siguientes formas normales, ya que para un buen diseño de una base de datos basta con normalizar hasta la tercera forma normal o la forma normal de Boyce-Codd (no siempre se puede normalizar hasta esta forma normal sin pérdida de información o de dependencias funcionales), pero sí es recomendable conocer al menos la definición de la cuarta forma normal.

En la mayoría de los escritos relacionados con la normalización de las Bases de Datos se recomienda no pasar de la tercera forma normal o llegar como mucho a la FNBC para que no haya pérdida de dependencias funcionales en las sucesivas descomposiciones que se realizan para alcanzar las distintas formas normales. **Alcanzar la 4FN o la 5FN es posible únicamente en un número muy reducido de casos, y siempre hay que observar con detenimiento que se conservan todas las dependencias funcionales que definen el sistema que estamos modelando.**

Para saber más

Para profundizar en la teoría de las dependencias multivaluadas y las formas normales 4ª y 5ª, entra en los siguientes enlaces donde encontrarás más información relacionada con el tema:

Dependencias multivaluadas **[Versión en caché]**

Formas normales **[Versión en caché]**

Más formas normales **[Versión en caché]**

Una relación R (X, Y, Z) está en cuarta forma normal si para cada dependencia multivaluada no trivial X -->--> Y, X es una clave primaria de R. Esto es equivalente a decir que R está en 4FN si está en FNBC y todas las dependencias multivaluadas en R son de hecho dependencias funcionales.

Dada una relación **R (X, Y, Z)** con **X, Y, Z** atributos simples o compuestos, y tal que en la relación se dan las siguientes dependencias multivaluadas:

X-->-->Y

X-->-->Z

Para transformar **R** en una relación que esté en **4FN**, dicha relación se descompone en otras dos relaciones de la siguiente manera:

- **R (X, Y)** con **X-->Y**
- **R' (X, Z)** con **X-->Z**

Las dos relaciones resultantes, **R** y **R'** sí están en **4FN**.

Veamos esta definición con un ejemplo concreto.

Consideramos la siguiente relación:

```
AGENDA (NOMBRE, TELEFONO, CORREO)
```

Es evidente que tenemos las siguientes dependencias funcionales:

```
NOMBRE -->--> TELEFONO
```

```
NOMBRE -->--> CORREO
```

por lo que **AGENDA** no está en **4FN**. Lo solucionamos considerando las siguientes relaciones:

```
R (NOMBRE, TELEFONO)
```

```
R' (NOMBRE, CORREO)
```

Ahora las relaciones resultantes **R** y **R'** sí están en **4FN**.

En el siguiente enlace tienes una animación sobre la relación existente entre unas formas normales y otras.

¿Qué relación tienen unas formas normales con otras?

Como las cosas se aprenden mejor practicándolas y un **ejemplo** vale más que mil palabras, vamos a ver la normalización "en acción", proporcionándote un ejemplo, y los pasos a seguir para

normalizar.

Consideramos la siguiente relación:

ALUMNO

DNI	NOMBRE	APELLIDO	C_POSTAL	PROVINCIA	ASIGNAT	PROFESOR	NOTA
12345678	Silvia	Thomas	18019	Granada	Informática	Alfonso Bonillo	9
12345678	Silvia	Thomas	18019	Granada	Matemáticas	Narciso Jáimez	7
34567890	Miguel A.	Pérez	14001	Córdoba	Matemáticas	Narciso Jáimez	8
34567890	Miguel A.	Pérez	14001	Córdoba	Lengua	Sebastián López	9
23456789	Diego	Rodríguez	04720	Almería	Lengua	Sebastián López	9
23456789	Diego	Rodríguez	04720	Almería	Informática	Alfonso Bonillo	9

¿Cumple esta relación con los criterios de las distintas formas normales? ¿Está en 2FN, en 3FN, en FNBC, en 4FN...?

Estudiemos una a una las distintas formas normales para saber si la relación **ALUMNO** las verifica, y en caso de no verificar alguna, vamos a ir solucionando el problema como hemos visto en la teoría en los apartados anteriores.

Primera y Segunda Forma Normal

- **¿Está la relación ALUMNO_MATRICULA en 1FN?**

Podemos decir que la relación **ALUMNO_MATRICULA está en 1FN** ya que no hay ningún atributo que no sea atómico, es decir, en cada tupla de la relación todos los atributos toman un único valor.

- **¿Está en 2FN?**

Para hacer esta comprobación tenemos que ver que todos los atributos que no forman parte de la clave primaria tienen dependencia funcional completa respecto de dicha clave. Empezamos antes de nada determinando cuál es la clave primaria de esta relación.

Está claro que para que cada tupla de la relación sea única, la clave primaria de la relación ALUMNO_MATRICULA tiene que estar compuesta por los atributos DNI y ASIGNAT.

Vemos las dependencias que se observan en la relación:

DNI à NOMBRE, APELLIDO, C_POSTAL, PROVINCIA
C_POSTAL à PROVINCIA
DNI, ASIGNAT à NOTA
ASIGNAT à PROFESOR (también podría ser PROFESOR --> ASIGNAT)

Luego **ALUMNO_MATRICULA** no está en **2FN**. ¿Cómo lo solucionamos

Descomponiendo la relación en otras que sí estén en **2FN** como hemos visto en apartados

anteriores. Para ello creamos nuevas relaciones de tal manera que en cada una de ellas tengamos una clave primaria y un conjunto de atributos que tengan dependencia funcional completa respecto de dicha clave. Teniendo en cuenta que la primera y la cuarta dependencia violan la 2FN, crearemos dos nuevas tablas y siguiendo este criterio nos quedan las siguientes relaciones:

ALUMNO_MATRICULA (DNI, NOMBRE, APELLIDO, C_POSTAL, PROVINCIA)
CALIFICACION (DNI, ASIGNAT, NOTA)
PROFESOR (ASIGNAT, PROFESOR)

Ahora sí que todas las relaciones **están en 2FN**.

Tercera Forma Normal y FNBC

- **¿Están las relaciones obtenidas en 3FN?**

Para comprobar si una relación está en 3FN tenemos que asegurarnos de que no existen dependencias funcionales transitivas en dicha relación, y vemos que con la descomposición que hemos realizado para conseguir que nuestra relación **ALUMNO** esté en 2FN hemos conseguido que las relaciones resultantes estén también en 3FN ya que tenemos las siguientes dependencias en las distintas relaciones:

```
ALUMNO ( DNI, NOMBRE, APELLIDO, C_POSTAL)
DNI --> NOMBRE, APELLIDO, C_POSTAL

C_POSTAL ( C_POSTAL, PROVINCIA)
C_POSTAL --> PROVINCIA

CALIFICACION (DNI, ASIGNAT, NOTA)
DNI, ASIGNAT --> NOTA

PROFESOR (ASIGNAT, PROFESOR)
ASIGNAT --> PROFESOR
```

Ninguna de las dependencias es transitiva por lo que todas las relaciones **se encuentran en 3FN**.

- **¿Están las relaciones en FNBC?**

Para realizar esta comprobación basta con analizar si en las relaciones todo determinante es clave candidata.

Podemos comprobar que en nuestro caso cada una de las relaciones tiene un único determinante que además coincide con la clave primaria de la relación, por lo que todas **las relaciones están en FNBC**.

La siguiente animación te presenta este proceso de forma gráfica:

Proceso de normalización de nuestro ejemplo

Partimos del siguiente supuesto: Queremos **normalizar** la siguiente información sobre una empresa de transporte y de la que tenemos un diseño inicial compuesto por una única tabla con la siguiente estructura:

```
ENVIO (cod_envio, matricula_camion, modelo, capacidad, cliente, direccion_cliente, pedido_cliente, articulo_pedido_cliente, volumen_articulo_cliente)
```

Está claro que este diseño no es el más óptimo para una base de datos. Tener todos los datos de una empresa en una única tabla no es la mejor manera de almacenar dicha información. Vamos a aplicar la teoría de la normalización para conseguir un diseño óptimo de la Base de Datos.

Primera y Segunda Forma Normal

- **¿Está en 1FN?**

Como en esta ocasión únicamente nos han dado la intención de la relación, suponemos que todos los valores de los atributos de la relación son atómicos, por lo que dicha relación **está en 1FN**.

- **¿Están en 2FN?**

Para hacer esta comprobación tenemos que ver que todos los atributos que no forman parte de la clave primaria tienen dependencia funcional completa respecto de dicha clave. También debemos tener en cuenta una máxima que debe regir en todo momento nuestro diseño: "**hechos distintos se deben almacenar en objetos distintos**". Máxima que no se cumple en nuestro caso, por lo que empezamos descomponiendo la relación en otras que sí la verifiquen.

Podemos considerar las siguientes relaciones con sus dependencias:

```
ENVIO (cod_envio, matricula_camion, modelo_camion, capacidad_camion)
cod_envio --> matricula_camion
matricula_camion --> modelo_camion, capacidad_camion

PEDIDO_CLIENTE (pedido_cliente, cliente, direccion_cliente, codigo_envio)
pedido_cliente --> cliente, codigo_envio
cliente --> direccion_cliente

PEDIDO_ARTICULO (pedido_cliente, codigo_articulo, volumen_articulo)
pedido_cliente --> codigo_articulo
codigo_articulo --> volumen_articulo
```

Ahora analizamos si cada una de las relaciones obtenidas está en **2FN**.

- **ENVIO**: Los atributos `modelo_camion` y `capacidad_camion` dependen únicamente del atributo `matricula_camion`, por lo que esta relación se descompone en las siguientes:
  - `ENVIO (cod_envio, matricula_camion)`
  - `CAMION (matricula_camion, modelo_camion, capacidad_camion)`
- **PEDIDO_CLIENTE**: El atributo direccion_cliente depende únicamente del atributo cliente, por lo que podemos descomponer esta relación como sigue:
  - `PEDIDO_CLIENTE (pedido_cliente, cliente, codigo_envio)`
  - `CLIENTE (cliente, direccion_cliente)`
- **PEDIDO_ARTICULO**: El atributo `volumen_articulo` depende únicamente del atributo `codigo_articulo`, por lo que podemos descomponer esta relación de la siguiente manera:
  - `PEDIDO_ARTICULO (pedido_cliente, codigo_articulo`
  - `ARTICULO (codigo_articulo, volumen_articulo)`

Las relaciones `ENVIO, CAMION, PEDIDO_CLIENTE, CLIENTE, PEDIDO_ARTICULO` y

ARTICULO resultantes sí están ahora en **2FN**.

Tercera Forma Normal y FNBC

- **¿Están las relaciones obtenidas en 3FN?**

¿Qué condición debe cumplir una relación para que esté en 3FN?

Que no tenga ninguna dependencia transitiva entre sus atributos. Si analizamos cada una de las relaciones que hemos obtenido en el apartado anterior podemos comprobar que no existe ninguna en la que aparezca una relación transitiva.

- **¿Están en FNBC?**

Para realizar esta comprobación basta con analizar si en las relaciones todo determinante es clave candidata.

Podemos comprobar que en esta ocasión, al igual que ocurrió en el ejemplo anterior, cada una de las relaciones tiene un único determinante que además coincide con la clave primaria de la relación, por lo que todas **las relaciones están en FNBC**.