



TEMA 3

DISEÑO LÓGICO. NORMALIZACIÓN

Bases de Datos
CFGs DAW

Autor: Raquel Torres
Revisado por: Francisco Aldarias
Revisado por: Pau Miñana
2020/2021

Licencia



Reconocimiento – NoComercial – CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:



Importante



Atención




Interesante

ÍNDICE DE CONTENIDO

1. Introducción.....	4
2. Dependencia funcional.....	5
2.1 Dependencia funcional.....	5
2.2 Dependencia funcional completa.....	5
2.3 Dependencia funcional transitiva.....	6
2.4 Diagrama de dependencias.....	6
3. Normalización.....	7
3.1 Primera Forma Normal - 1FN.....	7
3.2 Segunda Forma Normal - 2FN.....	8
3.3 Tercera Forma Normal - 3FN.....	11
4. Ejemplo completo de normalización.....	12

UD3 DISEÑO LÓGICO. NORMALIZACIÓN

1. INTRODUCCIÓN

 La **normalización** es un proceso de refinamiento para comprobar la calidad de nuestro modelo verificando que las relaciones o tablas del modelo Relacional obtenido no tienen redundancias ni inconsistencias.

Posiblemente las primeras veces nuestro modelo Relacional cambie algo al ser normalizado, pero poco a poco os daréis cuenta que al ir creando vuestro modelo de datos, inconscientemente, ya estaréis aplicando la normalización a lo largo del proceso y al final, al comprobar si está normalizado, comprobaréis que cumple todas las reglas necesarias.

El proceso de normalización consiste en la aplicación de una serie de reglas que nos sirven para verificar, y en algunas ocasiones modificar, nuestro modelo Relacional.

Para ello, vamos a aplicar las siguientes fases de la normalización (existen más fases pero se salen de los objetivos de este curso):

- Primer forma normal - 1FN
- Segunda forma normal - 2FN
- Tercera forma normal - 3FN

El proceso de normalización se realizará de la siguiente forma, primero comprobaremos que todas nuestras relaciones o tablas están en 1FN y si no es así realizaremos los cambios necesarios para que así sea. Una vez lo tenemos todo en 1FN aplicaremos las reglas para ver si están en 2FN y así sucesivamente. De tal forma que si decimos que nuestro modelo está en 3FN ya sabemos que cumple 1FN, 2FN y 3FN.

Antes de comenzar a ver las reglas de la normalización debemos conocer el concepto de dependencia funcional y sus tipos.

2. DEPENDENCIA FUNCIONAL

2.1 Dependencia funcional

☞ Un atributo B depende funcionalmente de otro atributo A si a **cada valor de A** le corresponde un **único valor de B**. Se dice que A implica B o lo que es lo mismo **$A \rightarrow B$** . En este caso A recibe el nombre de **implicante**.

Por ejemplo en la siguiente tabla *Personas* podemos tener:

PERSONAS (DNI, Nombre, Fecha_Nacimiento);

Podemos decir que $DNI \rightarrow Nombre$, ya que un DNI se corresponde con un único Nombre, es decir, a través del DNI podemos localizar el nombre de la persona a la que pertenece.

Luego cuando tenemos un atributo A que implica B ($A \rightarrow B$) es lo mismo que decir que B depende funcionalmente de A.

2.2 Dependencia funcional completa

☞ Dado un conjunto de atributos A formado por a_1, a_2, a_3, \dots (estos atributos formarán una clave primaria compuesta) existe una dependencia funcional completa cuando B depende de A pero no de un subconjunto de A.

Es decir que para obtener B son necesarios todos los elementos del conjunto A, o dicho de otra forma B depende de todos los atributos que forman la clave primaria A y no puede depender solamente de parte de ellos.

Por ejemplo cuando tenemos una tabla de *Compras* de la siguiente forma:

COMPRAS (Referencia_Producto, Código_Proveedor, Dirección_Proveedor, Cantidad, Precio)

En este caso nuestra clave primaria está formada por dos campos, la *Referencia_Producto* y el *Código_Proveedor*. Pues bien, debemos comprobar si todos los demás atributos tienen una dependencia funcional de toda la clave primaria.

Podemos observar que la *Dirección_Proveedor*, no dependerá de ambos, sino solamente del *Código_Proveedor*, luego para ese atributo **no** existiría dependencia funcional completa. Sin embargo el resto de los campos, tanto la *Cantidad*, como el *Precio* acordado dependen del conjunto, es decir, dependen completamente de la clave primaria, luego aquí sí tenemos una dependencia funcional completa.

2.3 Dependencia funcional transitiva

Este tipo de dependencia implica a tres atributos. Cuando existe una dependencia $A \rightarrow B$ y a su vez tenemos que $B \rightarrow C$, podemos decir que existe una dependencia funcional transitiva entre A y C.

Por ejemplo, si tenemos una tabla de *Productos* como la siguiente:

PRODUCTOS (Referencia_Producto, Nombre, Precio, Stock, Fabricante, País);

Aquí podemos encontrar que el *Fabricante* depende de la *Referencia_Producto*, es decir:

$\text{Referencia_Producto} \rightarrow \text{Fabricante}$

Y por otro lado el *País* también puede depender del *Fabricante*, luego:

$\text{Fabricante} \rightarrow \text{País}$.

Con estas dos dependencias podemos afirmar que el *País* **depende transitivamente** de la *Referencia_Producto*.

2.4 Diagrama de dependencias

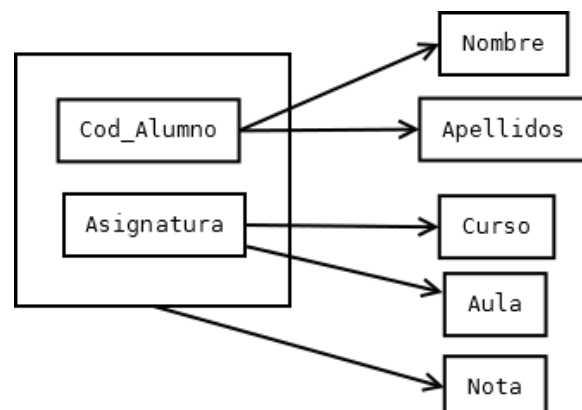
Los diagramas de dependencias muestran todos los atributos de una relación y sus dependencias. Habitualmente se encierran en una caja el conjunto de atributos de la clave primaria y se unen mediante flechas todos los atributos que muestren alguna dependencia entre ellos y también con el conjunto de claves. Resultan muy útiles a la hora de normalizar, puesto que permiten analizar si existe una dependencia funcional completa o dependencias transitivas. Veamos el siguiente ejemplo, aunque funcionalmente no tenga mucho sentido:

Supongamos que tenemos una relación **Alumno** en la que representamos los datos de los alumnos y las notas de cada una de las asignaturas en que está matriculado. La CP es el código del alumno y la asignatura:

Alumno (cod_alumno, nombre, apellidos, asignatura, nota, curso, aula)

Es obvio que no todos los atributos dependen completamente de la CP. Veamos las dependencias funcionales de cada uno de los atributos con respecto a los atributos de la clave:

- $\text{cod_alumno} \rightarrow \text{nombre}$
- $\text{cod_alumno} \rightarrow \text{apellidos}$
- $\text{asignatura} \rightarrow \text{curso}$
- $\text{asignatura} \rightarrow \text{aula}$
- $\{\text{cod_alumno}, \text{asignatura}\} \rightarrow \text{nota}$



Los apellidos y el nombre sólo dependen del código del alumno (no confundir aquí una relación transitiva con nombre y apellidos pues un mismo nombre no implica unos apellidos concretos).

El curso, al igual que el aula, sólo depende de la asignatura, independientemente de los alumnos, notas o nombres. A no ser que queramos asignar un aula al curso, que no es el caso, tampoco dependen directamente entre ellas

La nota sí que depende tanto del código del alumno como de la asignatura, ya que cada alumno tendrá una nota propia en cada una de las asignaturas que curse


Posteriormente veremos que hacer con esta información.

3. NORMALIZACIÓN

Vamos a ver cómo aplicar las formas normales hasta conseguir que nuestro Modelo Relacional esté normalizado¹.

3.1 Primera Forma Normal – 1FN.

Es una forma normal inherente al esquema relacional, por lo que su cumplimiento es obligatorio; es decir toda tabla realmente relacional la cumple.

 Se dice que una tabla se encuentra en **primera forma normal** si y sólo si los **valores** que componen cada **atributo** de una tupla son **atómicos y no derivados**, es decir, cada atributo de la relación toma un único valor del dominio correspondiente y no hay valores definidos en función de otros atributos

Para eliminar los valores derivados se sustituyen por los atributos de los que dependen si es necesario; por ejemplo el atributo *edad* suele ser derivado de la *fecha de nacimiento*, puesto que si es un valor fijo su validez tiene un periodo de tiempo limitado a un año. Así pues eliminamos dicho atributo siempre que la fecha de nacimiento esté almacenada ya que podemos recuperar su valor.

Supongamos ahora que tenemos la siguiente tabla **Centros**.

Código_Centro	Nombre	Localidad	Teléfono
45005467	IES Ribera del Tajo	Talavera de la Reina	925722233 - 925722804
45005239	IES Azarquiel	Toledo	925267843 – 925637843
36003748	IES El Plantío	Huelva	973847284

1Nota: Después de normalizar nuestro diseño, en algunas ocasiones se desnormaliza por cuestiones de rendimiento o por simplificación de algunas consultas complejas, pero eso no implica que no haya que normalizar. Es decir, primero se normaliza y después si las circunstancias lo requieren, cambiaremos lo que sea necesario. Pero ese cambio es controlado ya que lo hacemos nosotros y podremos acotar las consecuencias que pueda tener.

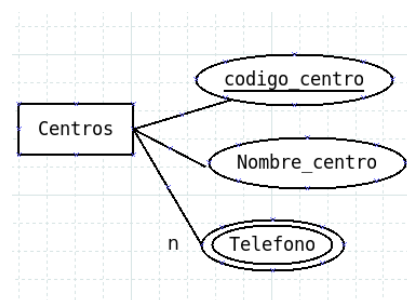
Podemos observar como en el campo teléfono hay ocurrencias en las que hay más de un valor. Este campo rompe la regla de la 1FN. La forma de resolverlo es creando una nueva tabla formada por el campo que contiene múltiples valores y la clave principal de la tabla. Quedando de la siguiente forma:

Nueva tabla **Centros**.

Código_Centro	Nombre	Localidad
45005467	IES Ribera del Tajo	Talavera de la Reina
45005239	IES Azarquiel	Toledo
36003748	IES El Plantío	Huelva

Y una nueva tabla que llamaremos **Teléfonos**.

Código_Centro	Teléfono
45005467	925722233
45005467	925722804
45005239	925267843
45005239	925637843
36003748	973847284



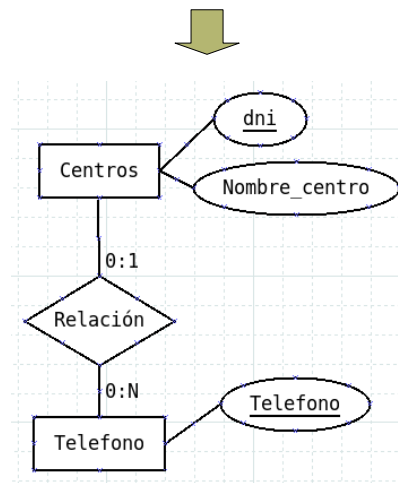
Centros (Código_Centro, Nombre, Localidad)

CP:{Código_Centro}

Teléfonos(Código_Centro, Teléfono)

CP:{Código_Centro, Teléfono}

CA:{ Código_Centro} → Centros



3.2 Segunda Forma Normal - 2FN.

Se dice que una relación se encuentra en **2FN** si y sólo si está en 1FN y **cada atributo** de la relación que no forma parte de la clave principal, **depende funcionalmente de la clave primaria completa**. La 2FN se aplica a las relaciones que tienen claves primarias compuestas por dos o más atributos. Si una relación está en 1FN y su clave primaria es simple, entonces también está en 2FN.

Es importante tener presente que las relaciones que no están en 2FN pueden sufrir anomalías cuando se realizan actualizaciones (inserciones, borrados o modificaciones).

Para pasar una relación que está en 1FN a 2FN hay que eliminar las dependencias

parciales de la clave primaria. Para ello, se eliminan los atributos que son parcialmente dependientes y se ponen en una nueva relación y como clave primaria la parte de la que dependen de la clave primaria. De esta manera tendremos dos tablas, una con los atributos que dependen de manera completa de la clave y otra tabla con los atributos que sólo dependen de una parte de la clave.

Supongamos que tenemos la siguiente tabla **Empleados**:

<u>Empleado</u>	<u>Especialidad</u>	Empresa
Juan Velasco	Bases de Datos	IBM
Juan Velasco	Sistemas Linux	IBM
Ana Comarce	Bases de Datos	Oracle
Javier Gil	Sistemas Windows	Microsoft
Javier Gil	Desarrollo .Net	Microsoft

Podemos ver que como clave principal se ha elegido el conjunto **Empleado + Especialidad**. Sin embargo podemos observar que el atributo Empresa no depende de toda la clave sino solo de parte de ella, en este caso de **Empleado**.

La forma de solucionarlo será separar en una tabla los campos que no dependen de toda la clave junto a la parte de la clave de la que dependen. En este caso será el campo **Empresa** y la nueva clave será **Empleado**.

El resultado será el siguiente:

Tabla **Empleados**:

<u>Empleado</u>	Empresa
Juan Velasco	IBM
Ana Comarce	Oracle
Javier Gil	Microsoft

Tabla **Lugares_Trabajo**:

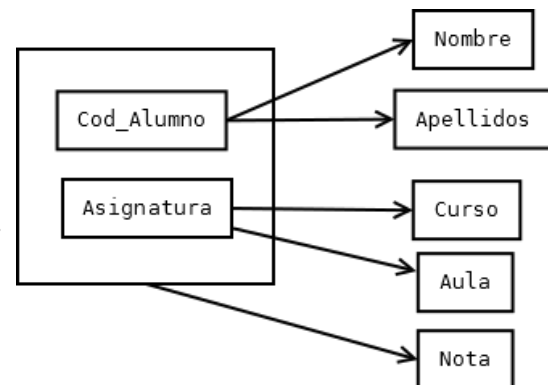
<u>Empleado</u>	<u>Especialidad</u>
Juan Velasco	Bases de Datos
Juan Velasco	Sistemas Linux
Ana Comarce	Bases de Datos
Javier Gil	Sistemas Windows
Javier Gil	Desarrollo .Net

Recordemos ahora el ejemplo que se ha expuesto en el apartado de la tabla de dependencias. Para asegurar su comprensión añadimos una tabla con datos de ejemplo que permiten comprobar las dependencias que se determinaron.

Alumno (cod_alumno, nombre, apellidos, asignatura, nota, curso, aula)

<u>cod_alumno</u>	nombre	apellidos	<u>asignatura</u>	nota	curso	aula
1111	Pepe	García	Lengua I	5	1	15
1111	Pepe	García	Inglés II	5	2	16
2222	María	Suárez	Inglés II	7	2	16
2222	María	Suárez	Ciencias II	7	2	14
3333	Juan	Gil	Plástica I	6	1	18
3333	Juan	Gil	Matemáticas I	6	1	12
4444	Francisco	Montoya	Lengua II	4	2	11
4444	Francisco	Montoya	Matemáticas I	6	1	12
4444	Francisco	Montoya	Ciencias I	8	1	14

Es obvio que no todos los atributos dependen completamente de la CP, como indica el diagrama de dependencias, así pues, vistas las dependencias funcionales es necesario crear tres relaciones:



1. Una para los atributos que dependen únicamente del Cod_Alumno (Alumnos).
2. Otra para guardar las asignaturas existentes (Asignaturas).
3. Una para los atributos que sí dependen funcionalmente de la CP anterior (Notas)

Alumnos(cod_alumno, nombre, apellidos)

Asignaturas(asignatura, curso, aula)

Notas(cod_alumno, asignatura, nota)

CAj: {cod_alumno} → Alumnos

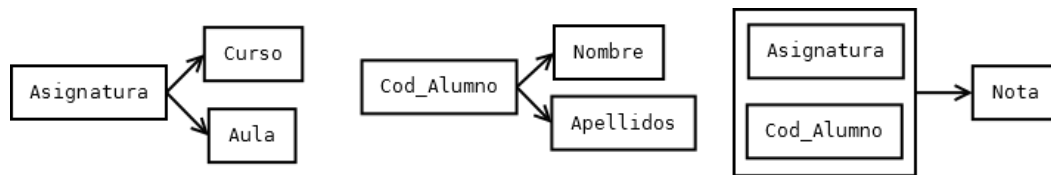
CAj: {asignatura} → Asignaturas

<u>cod_alumno</u>	<u>asignatura</u>	nota
1111	Lengua I	5
1111	Inglés II	5
2222	Ciencias II	7
2222	Inglés II	7
3333	Plástica I	6
3333	Matemáticas I	6
4444	Lengua II	4
4444	Matemáticas I	6
4444	Ciencias I	8

<u>asignatura</u>	curso	aula
Lengua I	1	15
Inglés II	2	16
Ciencias II	2	14
Plástica I	1	18
Matemáticas I	1	12
Lengua II	2	11

<u>cod_alumno</u>	nombre	apellidos
1111	Pepe	García
2222	María	Suárez
3333	Juan	Gil
4444	Francisco	Montoya

Ahora sí quedarían 3 diagramas funcionalmente dependientes



3.3 Tercera Forma Normal - 3FN.

Una tabla está en 3FN si está en 2FN y los **atributos** que **no** forman parte de la **clave no dependen** de **otros atributos que no son clave**, es decir no existen dependencias transitivas de la clave.

Las Claves alternativas están exentas de esta limitación, puesto que también podrían ser claves. Por tanto, los otros atributos pueden ser funcionalmente dependientes de una Clave alternativa (o conjunto) sin romper 3FN.

O dicho de otra forma, los atributos de la relación no dependen unos de otros, dependen únicamente de la clave, ya sea simple o compuesta.

Supongamos que un club de tenis ha creado una base de datos para guardar los campeones de los torneos en los que participan.

Tabla **Torneos**:

<u>Torneo</u>	<u>Año</u>	<u>Ganador</u>	<u>Fecha_Nacimiento_Ganador</u>
Alcores	2008	Javier Gil	20 de Abril de 1990
Estoril	2008	Alberto Sanz	15 de Julio de 1991
Getafe	2008	Jacinto Martin	10 de Enero de 1989
Santa María	2008	Roberto Loaisa	25 de Febrero de 1989
Alcores	2009	Jacinto Martin	10 de Enero de 1989
Estoril	2009	Ignacio Gómez	20 de Septiembre de 1990
Lisboa	2009	Roberto Loaisa	25 de Febrero de 1989
Getafe	2009	Roberto Loaisa	25 de Febrero de 1989
Santa María	2009	Javier Gil	20 de Abril de 1990

Podemos observar que el atributo *Ganador* depende del nombre del *Torneo* y del *Año* del mismo (ambas forman la clave primaria), sin embargo la fecha de nacimiento del ganador no depende de la clave principal, sino que depende del ganador. Aquí podemos ver una dependencia transitiva de la clave principal:

Torneo + Año → Ganador

Ganador → Fecha_Nacimiento_Ganador

Luego, *Fecha_Nacimiento_Ganador* depende transitivamente de la clave primaria.

Para que la tabla se encuentre en 3FN debemos evitar esta dependencia transitiva, para ello separaremos en una tabla diferente los campos que dependan transitivamente de la clave junto al campo del que dependen funcionalmente que actuará como clave primaria en la nueva tabla. En este caso quedará la tabla **Torneos** sin el atributo *Fecha_Nacimiento_Ganador*.

Nueva tabla **Torneos**:

<u>Torneo</u>	<u>Año</u>	<u>Ganador</u>
Alcores	2008	Javier Gil
Estoril	2008	Alberto Sanz
Getafe	2008	Jacinto Martín
Santa María	2008	Roberto Loaisa
Alcores	2009	Jacinto Martín
Estoril	2009	Ignacio Gómez
Lisboa	2009	Roberto Loaisa
Getafe	2009	Roberto Loaisa
Santa María	2009	Javier Gil

Y por otro lado se genera la tabla **Ganadores** donde la clave principal será **Ganador**:

<u>Ganador</u>	<u>Fecha_Nacimiento_Ganador</u>
Javier Gil	20 de Abril de 1990
Alberto Sanz	15 de Julio de 1991
Jacinto Martín	10 de Enero de 1989
Roberto Loaisa	25 de Febrero de 1989
Ignacio Gómez	20 de Septiembre de 1990

Además de esta forma normal, existen otras más cuyo estudio es muchas veces más teórico que práctico, es decir que se pueden demostrar matemáticamente pero que prácticamente no se suelen aplicar al crear un diseño real, por ello no las vamos a tratar en este curso.

4. EJEMPLO COMPLETO DE NORMALIZACIÓN

Cliente (numcli, nif, nombre, dir, nom_ciudad, nom_prov, telf, {cuenta}ⁿ)

({cuenta}ⁿ usaremos esta notación para expresar atributos compuestos {} y multivaluados ⁿ)

Cuenta (num_cuenta, tipo_c, saldo, {tarjeta}ⁿ)

El atributo *tarjeta* también es compuesto y multivaluado:

tarjeta(num_tarjeta, tipo_t, comision, limite)

Comentar que en este caso los atributos compuestos no han sido descompuestos en atributos simples. Si se hubiera hecho, aquí nos ahorraríamos algunos pasos.

Paso a 1FN

Lo primero es eliminar los atributos multivaluados y los atributos compuestos. La relación **Cliente** sólo tiene un atributo multivaluado llamado *cuenta*. Para ello quitamos el atributo de la tabla principal, buscamos la clave primaria y añadimos las restricciones:

Cliente(numcli, nif, nombre, dir, nom_ciudad, nom_prov, telf)
CP: {numcli}
UNIQUE: {nif}

A continuación creamos una nueva tabla llamada **Cuenta** para tener en cuenta sus múltiples valores:

Cuenta(numcli, {cuenta})

Eliminamos el atributo compuesto dividiéndolo en sus atributos simples:

Cuenta(numcli, num_cuenta, tipo_c, saldo, {tarjeta}ⁿ)

Volvemos a tener otro atributo multivaluado que hay que eliminar llamado *tarjeta*. Primero lo quitamos de la tabla anterior:

Cuenta(numcli, num_cuenta, tipo_c, saldo)

La relación ya no tiene ni atributos multivaluados ni compuestos así que buscamos la clave primaria y añadimos las restricciones.

Cuenta(numcli, num_cuenta, tipo_c, saldo)
CP: {num_cuenta}
CAj: {numcli} → Cliente

A continuación creamos otra tabla **Tarjeta**, con el num_cuenta asociado y sus atributos:

Tarjeta(num_cuenta, num_tarjeta, tipo_t, comisión, límite)

La relación ya no tiene ni atributos multivaluados ni compuestos así que, de nuevo, buscamos la clave primaria y añadimos las restricciones:

Tarjeta(num_cuenta, num_tarjeta, tipo_t, comisión, límite)
CP: {num_tarjeta}
CAj: {num_cuenta} → Cuenta

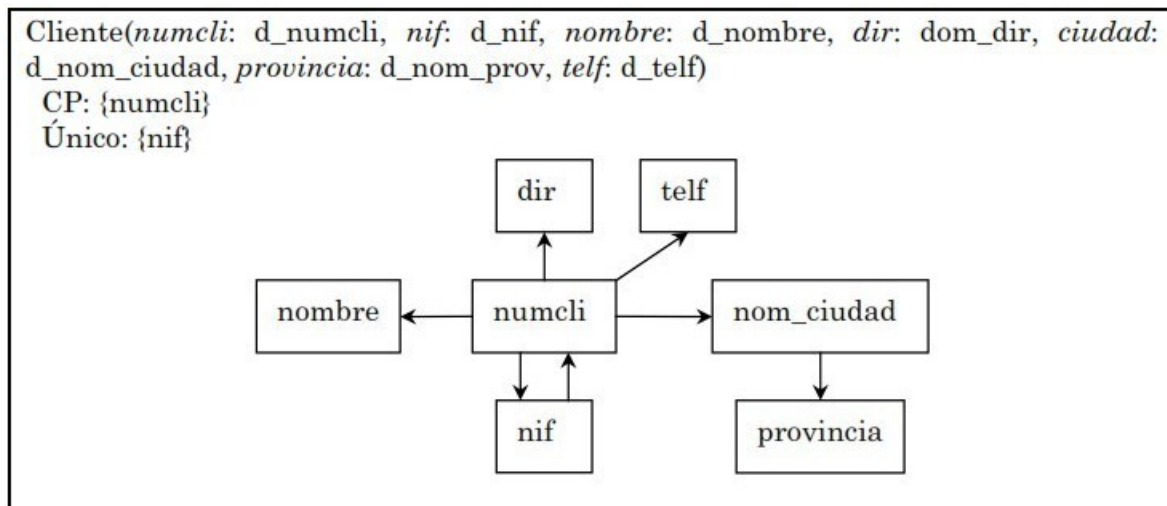
Las relaciones **Cliente**, **Cuenta** y **Tarjeta** ya se encuentran en 1FN.

Paso a 2FN

La 2FN obliga a que todos los atributos que no forman parte de la clave, dependan **completamente** de esta. Como ninguna de las relaciones obtenidas en el paso anterior tiene una clave compuesta, todas ellas se encuentran ya en 2FN.

Paso a 3FN

Ahora queda eliminar las dependencias transitivas y entre atributos. Veamos primero las dependencias de cada una de las tablas:



Analicemos las dependencias anteriores:

- Los atributos *nombre*, *dir*, *telf* y *nom_ciudad* dependen de *numcli* ya que el número de un cliente determina un único valor para los atributos anteriores.
- Con el *nif* pasa exactamente la misma dependencia, pero también sucede que *numcli* depende de *nif* y viceversa. Esto nos indica si no lo habíamos visto ya que el atributo *nif* es una clave alternativa y por tanto no supone ningún problema.
- Sin embargo, *provincia* depende de *nom_ciudad* ya que el nombre de una ciudad determina la provincia donde se encuentra (pero no al revés, ya que una provincia tiene muchas ciudades).

Hemos encontrado pues una relación transitiva que hay que eliminar.

Ciudad(nom_ciudad, nom_prov)

CP: {nom_ciudad}

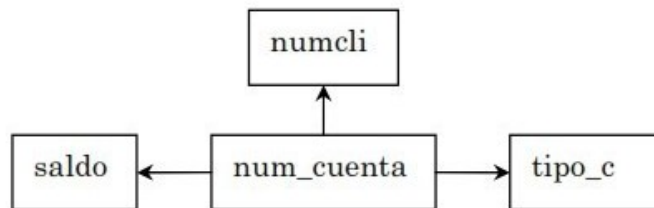
Cliente(numcli, nif, nombre, dir, nom_ciudad, telf)

CP: {numcli}

CAIt: {nif}

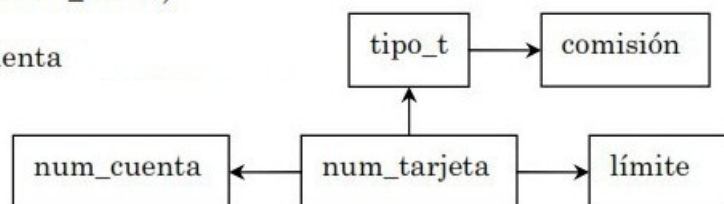
CAj: {nom_ciudad} → Ciudad

Cuenta(numcli: d_num_cli, num_cuenta: d_num_cuenta, tipo_c: d_tipo_c, saldo: d_saldo)
 CP: {num_cuenta}
 CA: {numcli} → Cliente



Esta relación no tiene relaciones transitivas entre atributos, ya se encuentra en 3FN.

Tarjeta(num_cuenta: d_num_cuenta, num_tarjeta: d_num_tarjeta, tipo_t: d_tipo_t, comisión: d_comisión, límite: d_límite)
 CP: {num_tarjeta}
 CA: {num_cuenta} → Cuenta



Analicemos las dependencias de la imagen:

- Los atributos *límite*, *tipo_t* y *num_cuenta* dependen de *num_tarjeta* ya que el número de una tarjeta determina el tipo de tarjeta, su límite establecido y la cuenta a la que va asociada.
- En cambio, el atributo *comisión* depende de *tipo_t* ya que la comisión que se cobra por una tarjeta dependerá del tipo de tarjeta.

Luego hemos encontrado una dependencia transitiva que hay que eliminar.

Comisión(comisión, tipo_t)

CP: {tipo_t}

Tarjeta(num_cuenta, num_tarjeta, tipo_t, límite)

CP: {num_tarjeta}

CAj: {num_cuenta} → Cuenta

CAj: {tipo_t} → Comisión

Así pues, el conjunto de relaciones definitivo en 3FN es el siguiente:

Ciudad(nom_ciudad, nom_prov)

CP: {nom_ciudad}

Cliente(numcli, nif, nombre, dir, nom_ciudad, telf)

CP: {numcli}

CAIt: {nif}

CAj: {nom_ciudad} → Ciudad

Cuenta(numcli, num_cuenta, tipo_c, saldo)

CP: {num_cuenta}

CAj: {numcli} → Cliente

Comisión(comisión, tipo_t)

CP: {tipo_t}

Tarjeta(num_cuenta, num_tarjeta, tipo_t, límite)

CP: {num_tarjeta}

CAj: {num_cuenta} → Cuenta

CAj: {tipo_t} → Comisión