
UD.3: MODELO LÓGICO.Parte 2

Normalización

Tema Extendido


Bases de Datos (BD)
CFGs DAM/DAW

Abelardo Martínez y Pau Miñana.
Basado y modificado de Sergio Badal y Raquel Torres.
Curso 2023-2024

ÍNDICE

- [1. ¿Qué es la normalización?](#)
- [2. Dependencias funcionales](#)
 - [2.1. Dependencia funcional](#)
 - [2.2. Dependencia funcional completa](#)
 - [2.3. Dependencia funcional transitiva](#)
 - [2.4. Diagrama de dependencias](#)
- [3. Normalización](#)
 - [3.1. Primera Forma Normal \(1FN\)](#)
 - [3.2. Segunda Forma Normal \(2FN\)](#)
 - [3.3. Tercera Forma Normal \(3FN\)](#)
 - [3.4. Otras formas normales](#)
- [4. Ejemplo completo de normalización](#)
 - [4.1. Primera forma normal \(1FN\)](#)
 - [4.2. Segunda forma normal \(2FN\)](#)
 - [4.3. Tercera forma normal \(3FN\)](#)
- [5. Bibliografía](#)

1. ¿Qué es la normalización?

 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. Este proceso, realizado correctamente, puede incluso corregir algunos fallos cometidos en el esquema E-R.

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 no encajan en los objetivos de este curso):

- Primera forma normal → **1FN**
- Segunda forma normal → **2FN**
- Tercera forma normal → **3FN**


El proceso de normalización se realizará de la siguiente forma:

1. Primero comprobaremos que todas nuestras relaciones o tablas están en 1FN y, si no es así, realizaremos los cambios necesarios para que así sea.
2. Una vez lo tenemos todo en 1FN aplicaremos las reglas para ver si están en 2FN y así sucesivamente.
3. Por último, 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. Dependencias funcionales

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**→**B**. En este caso **A** recibe el nombre de **implicante**.

Cuando tenemos un atributo A que implica B ($A \rightarrow B$) es lo mismo que decir que B depende funcionalmente de A. Veamos un ejemplo:

Persona (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.

2.2. Dependencia funcional completa



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

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

Evidentemente, si una clave principal no es compuesta sino solamente un campo, la dependencia funcional del resto de los campos con esta es completa, no necesita comprobación.

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

Compra (Producto, Vendedor, Dirección_V, Cantidad, Precio)

- En este caso la clave primaria está formada por dos campos, las referencias al *Producto* y el *Vendedor*. Podemos comprobar si todos los demás atributos tienen una dependencia funcional completa de la CP, ya veremos más adelante que esto es necesario para optimizar el funcionamiento de una BD y evitar redundancias.
- Podemos observar que la *Dirección_V*, no depende de ambos, sino solamente del *Vendedor*, luego para ese atributo no existiría dependencia funcional completa.
- Sin embargo, el resto de los campos -tanto la *Cantidad*, como el *Precio* acordado- dependen de todos los campos de la clave, así que sí que existe 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 (Ref, Nombre, Precio, Stock, Fabricante, País)

- Todos los campos dependen de la clave *Ref*, por ejemplo: $Ref \rightarrow Fabricante$.
- Por otro lado, el *País* depende del *Fabricante*, luego: $Fabricante \rightarrow País$.
- Así pues, aunque obviamente $Ref \rightarrow País$, esta dependencia *es transitiva*.

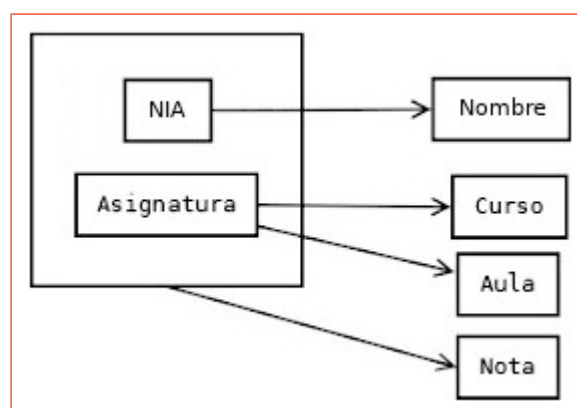
2.4. Diagrama de dependencias

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.

Supongamos una relación *Alumno* en la que se representan los datos de los alumnos y las notas de cada una de las asignaturas en que está matriculado.

Alumno (NIA, Nombre, 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:



- NIA → Nombre
- Asignatura → Curso
- Asignatura → Aula
- {NIA, Asignatura} → Nota


Así pues, tenemos que:

- El *nombre* solo depende del *NIA* del alumno.
- El *curso*, al igual que el *aula*, solo 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 *NIA* del alumno como de la *asignatura*, ya que cada alumno tendrá una nota propia en cada una de las asignaturas que curse.

Posteriormente veremos qué hacer con esta información.


3. Normalización

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

 Después de normalizar el diseño, en algunas ocasiones se **desnormaliza** algún aspecto por cuestiones de rendimiento, uso 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.

3.1. Primera Forma Normal (1FN)

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

 Se dice que una tabla se encuentra en **primera forma normal (1FN)** 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. De este modo, eliminamos dicho atributo siempre que la fecha de nacimiento esté almacenada, ya que podemos recuperar su valor.

¿Cómo realizar la transformación?

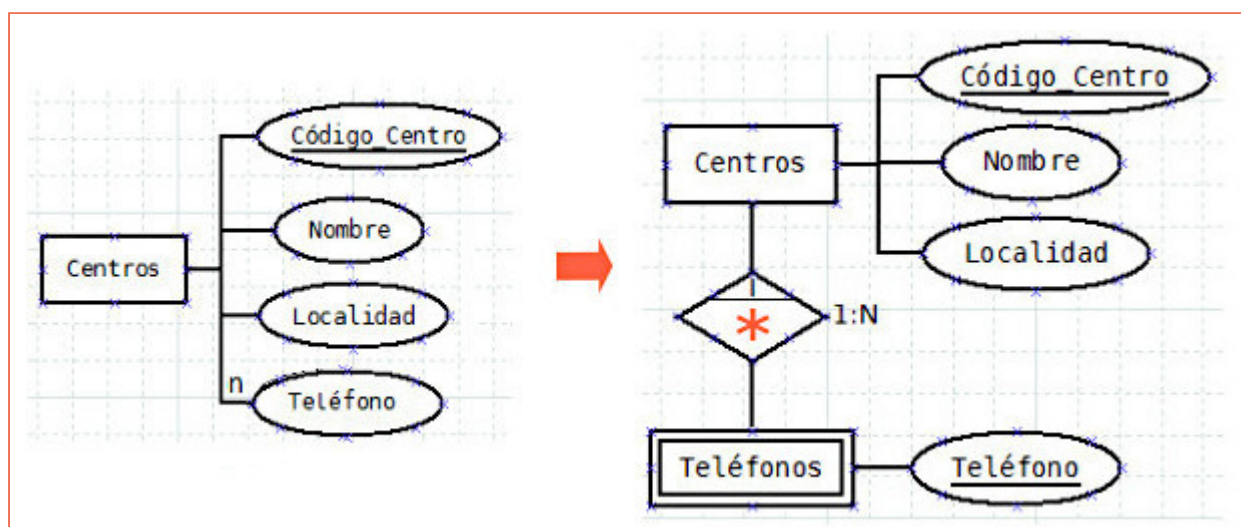
- Los atributos derivados se sustituyen por los atributos de los que dependen si es necesario.
- Los atributos compuestos se dividen en tantos campos como tenga la composición.
- Los atributos multivaluados se sustituyen por una relación en ER y se convierten a tablas como proceda.

EJEMPLO

Supongamos ahora que tenemos la siguiente tabla *Centros*.

<u>Cód_Centro</u>	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

Podemos observar como en el campo teléfono hay ocurrencias en las que hay más de un valor (multivaluados). 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. La solución se expone a continuación.



* En realidad se puede resolver con los 2 tipos de debilidad

- **Identificación:** Un teléfono puede pertenecer a varios centros.
- **Existencia:** Los números de teléfono son exclusivos de cada centro

Centros (Código_Centro, Nombre, Localidad)

CP: Código_Centro

Teléfonos (Código_Centro, Teléfono)

CP: {Código_Centro, Teléfono}

CAj: Código_Centro → Centros {Código_Centro}

Nueva tabla *Centros*:

<u>Código_Centro</u>	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*:

<u>Código_Centro</u>	<u>Teléfono</u>
45005467	925722233
45005467	925722804
45005239	925267843
45005239	925637843
36003748	973847284

En este caso, y solo para poder ver mejor como funciona, se ha dejado la opción en la que el teléfono, a parte de ser multivaluado, puede ser compartido entre varios centros. Obviamente en el contexto de este ejemplo la otra opción es más apropiada y la debilidad debería ser sólo de existencia (la clave de la nueva tabla, sería simplemente *Teléfono*).

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 contener información redundante y sufrir anomalías cuando se realizan actualizaciones (inserciones, borrados o modificaciones).

¿Cómo realizar la transformación?

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 en la tabla y se mueven a una tabla donde la clave principal sea sólo el o los atributos de los que dependen. Si una tabla con esa clave ya existe, se mueven a ésta, en caso contrario se crea una tabla nueva con esa CP.

EJEMPLO 1

Supongamos 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 se detalla a continuación.

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

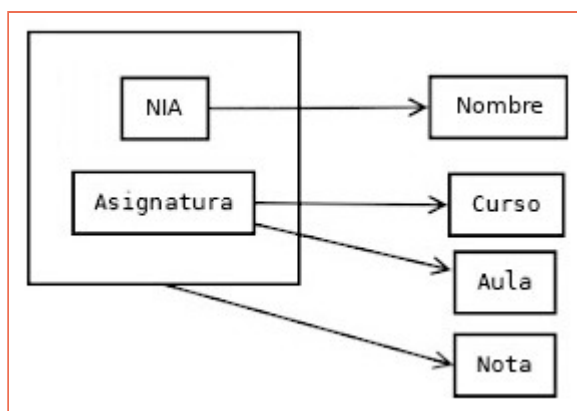
EJEMPLO 2

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 (NIA, Nombre, Asignatura, Nota, Curso, Aula)

<u>NIA</u>	Nombre	<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	8	2	14
3333	Juan Gil	Plástica I	6	1	18
4444	Iñigo Montoya	Lengua II	4	2	11
4444	Iñigo Montoya	Matemáticas I	6	1	12
4444	Iñigo Montoya	Ciencias II	8	2	14

No todos los atributos dependen de la CP completa, como indicaba su diagrama:



Así pues, vistas las dependencias funcionales es necesario crear tres relaciones:

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

Alumnos (NIA, Nombre)

Asignaturas (Asignatura, Curso, Aula)

Notas (NIA, Asignatura, Nota)

CAj: NIA → Alumnos {NIA}

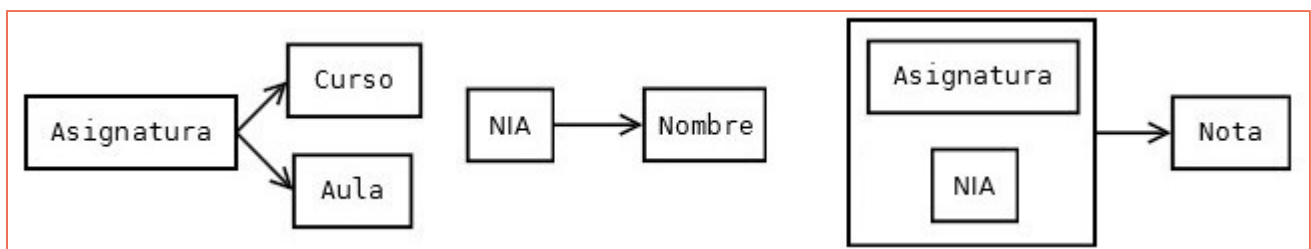
CAj: Asignatura → Asignaturas {asignatura}

<u>NIA</u>	Nombre
1111	Pepe García
2222	María Suárez
3333	Juan Gil
4444	Iñigo Montoya

<u>Asignatura</u>	Curso	Aula
Lengua I	1	15
Inglés II	2	16
Ciencias II	2	14
Plástica I	1	18
Lengua II	2	11
Matemáticas I	1	12

<u>NIA</u>	<u>Asignatura</u>	Nota
1111	Lengua I	5
1111	Inglés II	5
2222	Inglés II	7
2222	Ciencias II	8
3333	Plástica I	6
4444	Lengua II	4
4444	Matemáticas I	6
4444	Ciencias II	8

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 de la clave transitivas. 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.

EJEMPLO

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>	Ganador	F_Nacimiento_Ganador
Alcores	2008	Juan Gil	20/04/1990
Estoril	2008	Pepe García	15/07/1991
Getafe	2008	María Suárez	10/01/1989
Santa María	2008	Iñigo Montoya	20/09/1990
Alcores	2009	Pepe García	15/07/1991
Estoril	2009	Iñigo Montoya	20/09/1990
Lisboa	2009	Pepe García	15/07/1991

- El atributo *Ganador* depende del nombre del *Torneo* y del *Año* del mismo (ambas forman la clave primaria).
- La *F_Nacimiento_Ganador* no depende directamente de la CP, depende en realidad del *Ganador*. Aquí podemos ver una dependencia transitiva:
 - *Torneo* + *Año* → *Ganador*
 - *Ganador* → *F_Nacimiento_Ganador*

Luego, *F_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 CP**. Luego la tabla *Torneos* pierde el atributo *F_Nacimiento_Ganador*.

Tablas definitivas para *Torneos* y *Ganadores* :

<u>Torneo</u>	<u>Año</u>	Ganador
Alcores	2008	Juan Gil
Estoril	2008	Pepe García
Getafe	2008	María Suárez
Santa María	2008	Iñigo Montoya
Alcores	2009	Pepe García
Estoril	2009	Iñigo Montoya
Lisboa	2009	Pepe García

<u>Ganador</u>	F_Nacimiento
Juan Gil	20/04/1990
Pepe García	15/07/1991
María Suárez	10/01/1989
Iñigo Montoya	20/09/1990

Al igual como sucede en 2FN, es posible que la tabla adecuada para mover la *F_Nacimiento_Ganador* ya exista y no se necesite crear una tabla, por ejemplo una tabla de *Tenistas*. Es más, puede que esta tabla ya almacenase las fechas de nacimiento y en ese caso sólo se necesita eliminar el campo de la tabla *Torneos*. En caso de que no se almacene la fecha de nacimiento de todos los *Tenistas* sino sólo de los *Ganadores*, crear la nueva tabla puede evitar la aparición de muchos valores nulos en caso de almacenar esa fecha en la tabla *Tenistas*, asumiendo que la gran mayoría no habrán ganado ningún torneo.

3.4. Otras formas normales

Además de estas formas normales, existen otras más, cuyo estudio es muchas veces más teórico que práctico. Se pueden demostrar matemáticamente, pero prácticamente no se suelen aplicar al crear un diseño real de BBDD, por ello **no las vamos a tratar en este curso**.

<https://picodotdev.github.io/blog-bitix/2018/02/las-6-plus-2-formas-normales-de-las-bases-de-datos-relacionales/>

4. Ejemplo completo de normalización

Supongamos el modelo lógico siguiente, basado en una BD para la gestión de un supermercado:

```
Cliente (n_cliente, nif*, nombre, {factura}n)  
UK: nif  
Producto (n_prod, nombre, ciudad_origen, provincia_origen)  
Factura (n_fac, dia, hora, total)  
total: suma de los totales de las líneas de la factura.  
Línea_F (n_lin, n_fac, n_prod, precio_ud, cantidad, total)  
total= cantidad * precio_ud  
CAj: n_fac → Factura  
CAj: n_prod → Producto
```

4.1. Primera forma normal (1FN)

Lo primero es eliminar los atributos multivaluados, compuestos y derivados.

La relación *Cliente* solo tiene un atributo multivaluado llamado *factura*. Puesto que una factura no puede pertenecer a varios clientes, la CP de la nueva tabla sería simplemente la factura (debilidad de existencia, no identificación) y el *n_cliente* como clave ajena. Pero si nos fijamos ya existe una relación *Factura* con esa clave, así que simplemente movemos la clave ajena allí sin crear otra tabla.

```
Cliente (n_cliente, nif*, nombre)
UK: nif
Factura (n_fac, n_cliente*, dia, hora, total)
CAj: n_cliente → Cliente
total: suma de los totales de las líneas de la factura.
```

Los campos *total* tanto en la *Factura* como en *Línea_F* son derivados y se pueden calcular a través de los otros datos en la BD, con lo que pueden eliminarse.

🗨 Aquí podría entrar ese proceso de **desnormalización** mencionado anteriormente. En principio, ambos campos *total* sólo necesitan calcularse una vez cuando se cree la línea y no van a ser constantemente actualizados, puesto que una *Factura* no cambia con el tiempo. Si además consideramos que los usuarios pueden consultar de manera usual sus *Facturas* y ordenarlas por su *total*, entonces esto nos daría pie a ignorar la normalización de alguno o ambos campos.

En este ejemplo se opta por una normalización estricta y eliminamos los campos. Con esto todas las relaciones se encuentran ya en 1FN.

```
Cliente (n_cliente, nif*, nombre)
UK: nif
Producto (n_prod, nombre, ciudad_origen, provincia_origen)
Factura (n_fac, n_cliente*, dia, hora)
CAj: n_cliente → Cliente
Línea_F (n_lin, n_fac, n_prod, precio_ud, cantidad)
CAj: n_fac → Factura
CAj: n_prod → Producto
```

⚠ Hay que tener cuidado de no "perder" los cambios hechos en el proceso de normalización en cada paso. Por eso se recomienda, sobretodo al principio, apuntar al final de cada fase todas las tablas del modelo y así tenerlas juntas cuando las consultemos en la siguiente fase.

4.2. Segunda forma normal (2FN)

2FN fuerza que los demás atributos dependan de la CP completa, luego solo las claves compuestas podrían no cumplirlo. Solo necesitamos comprobar *Línea_F*.

Tanto el *n_prod* como la *cantidad* (incluso el *total* si no lo eliminamos) dependen de la CP completa, son propios de cada fila de *Línea_F*. En cambio el *precio_ud* no depende de la CP, sino del *Producto* (*n_prod*), así que lo movemos a esa tabla.

```
Cliente (n_cliente, nif*, nombre)
UK: nif
Producto (n_prod, nombre, precio, ciudad_or, provincia_or)
Factura (n_fac, n_cliente*, dia, hora)
CAj: n_cliente → Cliente
Línea_F (n_lin, n_fac, n_prod, cantidad)
CAj: n_fac → Factura
CAj: n_prod → Producto
```

Otro argumento a favor de la **desnormalización** en este caso. Puesto que ahora el *precio* está en la tabla de *Productos*, en cuanto se actualicen los *precios* (cosa que ocurrirá con el tiempo), el cálculo de totales para *Facturas* y *Líneas* solo podrán usar los precios actuales. Esto es nefasto para las *facturas ya existentes*, donde los valores reales se perderían. Consecuentemente, se decide conservar los campos calculados *total*, forzando que sean calculados solo en el momento de introducir la información y no actualizados automáticamente. El campo *precio_ud* puede moverse sin pérdida igualmente, ya que aún se podrá conocer el *precio* original de cada *Línea_F* dividiendo el *total* por la *cantidad*.

```
Cliente (n_cliente, nif*, nombre)
UK: nif

Producto (n_prod, nombre, precio, ciudad_or, provincia_or)

Factura (n_fac, n_cliente*, dia, hora, total)
CAj: n_cliente → Cliente
total: suma totales de líneas de la factura.

Línea_F (n_lin, n_fac, n_prod, cantidad, total_lin)
total_lin= cantidad * precio → (nprod → Producto {n_prod})
Calculado solo al momento de introducir la Línea,
no se actualiza automáticamente.
CAj: n_fac → Factura
CAj: n_prod → Producto
```


4.3. Tercera forma normal (3FN)

Ahora queda eliminar las dependencias transitivas y entre atributos. Veamos primero las dependencias de cada una de las tablas. Se debe hacer un examen exhaustivo, si es necesario mediante diagramas de dependencia de cada relación.

En este caso procedemos directamente a señalar las dependencias transitivas:

- En *Cliente* el *nombre* depende de la clave transitivamente a través del *nif*, pero puesto que este es claramente una clave alternativa, se ignora y no se cambia nada.
- Conviene no confundir la dependencia de *total_lin* como transitiva, aunque el campo sea calculado y tenga relación con el *n_prod*, depende directa y completamente de la clave de *Línea_F*.
- En *Producto*, la *provincia_or* depende transitivamente a través de la *ciudad_origen*, ya que una ciudad siempre pertenece a una provincia concreta, con lo que eliminamos la *provincia_or* en *Producto* e implementamos esa relación.

Cliente (n_cliente, nif*, nombre)

UK: nif

Ciudad (nom_ciudad, provincia)

Producto (n_prod, nombre, precio, ciudad_origen)

CAj: ciudad_origen → Ciudad {nom_ciudad}

Factura (n_fac, n_cliente*, dia, hora, total)

CAj: n_cliente → Cliente

total: suma totales de líneas de la factura.

Línea_F (n_lin, n_fac, n_prod, cantidad, total_lin)

total_lin= cantidad * precio → (nprod → Producto {n_prod})

Calculado solo al momento de introducir la Línea,
no se actualiza automáticamente.

CAj: n_fac → Factura

CAj: n_prod → Producto

Con esto, el diseño ya se encuentra en 3FN y está listo para su paso al Modelo Físico.

5. Bibliografía

- Iván López, M.^a Jesús Castellano. John Ospino. Bases de Datos. Ed. Garceta, 2a edición, 2017. ISBN: 978-8415452959
- Matilde Celma, Juan Carlos Casamayor y Laura Mota. Bases de datos relacionales. Ed. Prentice-Hall, 2003
- Cabrera Sánchez, Gregorio. Análisis y diseño detallado de aplicaciones informáticas de gestión. Ed. McGraw-Hill, 1st edition, 1999. ISBN: 8448122313
- Carlos Manuel Martí Hernández. Bases de dades. Desenvolupament d'aplicacions multiplataforma i Desenvolupament d'aplicacions web. Creative Commons. Departament d'Ensenyament, Institut Obert de Catalunya. Dipòsit legal: B. 12715-2016. <https://ioc.xtec.cat/educacio/recursos>