Normalización

¡Comencemos!

¿Qué aprenderás?

- Identificar y manejar grupos repetitivos para iniciar un proceso de normalización.
- Explicar cuando es necesario implementar la normalización entre tablas.
- Identificar las primeras tres formas normales para realizar una normalización a la base de datos.
- Reconocer la importancia de la desnormalización para facilitar consultas a la base de datos.
- Implementar técnicas de normalización para reducir redundancia entre tablas.

Introducción

En la industria podemos encontrar bases de datos enormes, que han sido creadas por un equipo completo de especialistas dedicados a la optimización del proceso de almacenamiento de información.

La mayoría de los problemas lógicos en las bases de datos se deben a las inconsistencias, a las malas relaciones o a la redundancia de información en los registros. En este capítulo aprenderás el concepto de normalización y su gran importancia para evitar esta redundancia de información innecesaria, además aprenderás cuándo se debe realizar un proceso inverso, para casos peculiares donde sea considerable tener datos almacenados repetidos en tablas diferentes con el objetivo de ofrecer la información en consultas menos procesadas.

La normalización es un proceso indispensable en bases de datos, en especial cuando empiezan a escalar por manipular grandes masas de información. La aplicación de las 3 formas normales será el camino en este capítulo para cumplir con el objetivo de cualquier base de datos.

Concepto de normalización

La normalización es un proceso que elimina la redundancia de una base de datos. El proceso consta de una serie de pasos en los cuales se van eliminando distintos tipos de redundancias con el propósito de prevenir inconsistencias. Estas etapas son muy importantes y reciben el nombre de formas normales. En este capítulo aprenderemos a llevar una base de datos a tercera forma normal.

¿Para qué sirve la normalización?

La normalización es una fórmula que podemos aplicar a una base de datos, a un modelo lógico o incluso a una tabla de excel o archivo con formato .csv que nos entreguen para crear finalmente una nueva base de datos sin redundancia.

Comúnmente se entiende por "normalizar una base de datos" pasarla a “tercera forma normal”, lo cual aprenderás más adelante en esta lectura. Sin embargo en cada etapa lo que estamos haciendo es precisamente normalizar.

En resumen, la normalización busca:

- Evitar la redundancia de datos.
- Simplificar la actualización de datos.
- Garantizar la integridad referencial (prevenir inconsistencia en las relaciones).

Para pasar una base de datos de una etapa a la otra tenemos que aplicar un conjunto de reglas, por lo cual definiremos algunos términos para luego entrar a normalizar.

Notación

Utilizaremos la siguiente notación para detallar la normalización de una tabla:

```
nombre_tabla(atributo_1, atributo_2, ..., atributo_n)
```

- Fuera de los paréntesis se tiene el nombre de la tabla con la que se trabajará.
- Dentro de los paréntesis se encuentran los atributos existentes.
- Hay casos en que dentro de una tabla pueden existir atributos repetidos. Para definirlos de forma explícita, se deben encapsular entre llaves { } de la siguiente forma:

```
nombre_tabla (
  atributo_1,
  atributo_2,
  ...,
  atributo_n,
  {atributo_repetido}
);
```

Te podrías estar preguntando ¿Atributos repetidos? ¿Cómo es posible? Veámoslo un caso a continuación.

¿Qué es un grupo repetitivo?

Lo más fácil es verlo con un ejemplo, tenemos la siguiente tabla Cliente:

N Cliente	Nombre	Teléfonos
1-9	Fernanda	123-456 123-457
2-7	Felipe	234-412
3-5	Francisco	123-411

Tabla 1. Grupo repetitivo tabla cliente.

En este caso, puede haber más de un teléfono asociado a cada cliente y no solo uno o dos, pueden haber cientos, no sabemos. Por eso es un atributo (o grupo de atributos) repetitivo, lo anterior es muy similar a esta nueva tabla:

N Cliente	Nombre	Teléfono1	Teléfono2
1	Fernanda	123-456	123-457
2	Felipe	234-412	
3	Francisco	123-411	

Tabla 2. Grupos repetitivos.

En ambos casos los atributos se repiten. Ahora si supiéramos que solo existe la posibilidad que el Cliente tenga 2 teléfonos, no sería un grupo repetitivo. El problema es cuando no conocemos realmente la cantidad de columnas como para poder almacenar los datos.

Utilizando la notación aprendida podríamos representar nuestra tabla como:

```
Cliente(N_Cliente, Nombre, {Teléfono})
```

Para estos casos debemos pensar en dividir esta entidad en 2 o más tablas y relacionarlas por llaves primarias y foráneas.

Identificando las claves primarias y foráneas

Para identificar la clave primaria dentro de una tabla, utilizaremos el símbolo # y letras en negrita, para representar claves foráneas dentro de una tabla las identificamos con letras cursivas. En caso de que sea necesario crear primarias compuestas, todos los atributos correspondientes a claves compuestas deberán estar en negritas y empezar con el signo numeral(#).

Ejemplo de clave primaria en una tabla:

```
Tabla1 (
  #clavePrimaria,
  atributo_1,
  ...,
  atributo_n
);
```

Ejemplo de clave primaria y foránea en una tabla:

```
Tabla2(
  #clavePrimaria,
  claveForanea,
  ...,
  atributo_n
);
```

Ejemplo de clave primaria compuesta:

```
Tabla3(
  #atributoPrimario,
  claveForanea,
  #otroAtributoPrimarioCompuesto,
  ...,
  atributo_n
);
```

Ahora que entendemos como incluir las claves a nuestras tablas en esta notación, apliquemoslo al ejemplo de la tabla cliente de la siguiente manera:

```
Cliente (
 #N_Cliente,
 Nombre,
 {telefono}
);
```

Como probablemente te das cuenta, por cada teléfono que tenga el cliente se tendrá que guardar otra fila en la tabla repitiendo los datos de la persona, esta redundancia la aprenderemos a tratar con la implementación de las formas normales a continuación.

Implementación de formas normales

Para ejemplificar el uso de las formas normales, utilizaremos el siguiente ejemplo, que corresponde a la factura de un paciente en un hospital, donde se muestran los datos del paciente y los ítems que consumió como verás en la imagen.

Factura		# 23464 20/04/2019
Nombre paciente: Marta Fuentes	#Paciente: 4724	
Dirección: Monjitas 245	Comuna: Santiago	
COD-SISTEMA-SALUD: 10	Ciudad: Santiago	

ITEM	NOMBRE	VALOR
200	Pieza semiprivada	150
204	Televisión	10
245	Rayos X	25
414	Exámenes	35
	SUBTOTAL	220
	%IMPUESTO	22
	TOTAL	242

Imagen 1. Ejemplo factura.

Realicemos un proceso de ingeniería inversa partiendo de la factura hasta el modelado de datos aplicando la normalización, pero antes debemos analizar la información que se nos presenta e identificar cuales son los atributos y grupos repetitivos en caso de existir, y lo haremos con la notación aprendida de la siguiente manera.

```
Factura(
 #numero_factura,
 fecha_factura,
 nombre_paciente,
 #numero_paciente,
 direccion,
 comuna,
 ciudad,
 #codigo_sistema_salud,
 {
  numero_item,
  nombre_item,
  valor_item
 },
 subtotal,
```

```
 impuesto,
 total
)
```

Los atributos subtotal, impuesto y total, se pueden eliminar, ya que estos atributos son derivables, es decir, **los podemos calcular** directamente en la aplicación o software, por lo que no sería necesario almacenarlo a menos que queramos persistir esta información para futuros filtros o estadísticas, no obstante también pudiéramos usar las funciones de las sentencias SQL como el "SUM()" para calcularlos, la situación entonces se convierte en un tema subjetivo que dependerá de la perspectiva del diseñador de bases de datos y en todo caso estos diseños pasan por un proceso de pruebas e incluso están enlazadas a la experiencia de usuario, considerando que más información solicitada significa más tiempo de espera en una consulta.

La tabla nos quedaría entonces como en la siguiente imagen.

#Factura	Fecha factura	Nombre paciente	Numero paciente	Dirección	Comuna	Ciudad	CSS	Numero Item	Nombre Item	Valor Item

Imagen 2. Tabla de atributos iniciales en factura.

Primera Forma Normal (1FN)

Ahora que tenemos la tabla Factura en la notación definida en puntos anteriores empecemos con la normalización.

Para que una tabla se encuentre normalizada acorde a la Primera Forma Normal (1FN), la tabla debe cumplir las siguientes condiciones:

- Cada campo o atributo deben ser atómicos, es decir debe contener un único valor.
- No pueden haber grupos repetitivos.

Como observamos en nuestro ejemplo, tenemos un grupo repetitivo de ***{numero_item, nombre_item,valor_item}***. Por lo que crearemos una nueva tabla llamada ***FacturaItem*** y su notación será la siguiente:

```
FacturaItem (
  #numero_factura,
  #numero_item,
  nombre_item,
  valor_item
)
```

Donde esta nueva tabla contiene todos los atributos del grupo repetitivo, y se crea una clave primaria compuesta con el ***numero_factura*** y ***numero_item***.

A la tabla factura se le elimina entonces el grupo repetitivo, quedando:

```
Factura(
  #numero_factura,
  fecha_factura,
  nombre_paciente,
  #numero_paciente,
  direccion,
  comuna,
  ciudad,
  #codigo_sistema_salud
)
```

El resultado de esta división la podemos ver gráficamente en la siguiente imagen.

Factura	# 23464 20/04/2019
Nombre paciente: Marta Fuentes	#Paciente: 4724
Dirección: Monjitas 245	Comuna: Santiago
COD-SISTEMA-SALUD: 10	Ciudad: Santiago

#Factura	ITEM	NOMBRE	VALOR
23464	200	Pieza semiprivada	150
23464	204	Televisión	10
23464	245	Rayos X	25
23464	414	Exámenes	35

Imagen 3. Primera división de la tabla factura.

Pasamos ahora la tabla factura de la izquierda a la misma presentación de la tabla de la derecha quedando como la imagen:

Factura

#Factura	Fecha factura	Nombre	#Paciente	Dirección	Comuna	Ciudad	CSS
23464	20/04/19	Marta Fuentes	4724	Monjtas 245	Santiago	Santiago	10

FacturaItem

#Factura	#Item	Nombre	Valor
23464	200	Pieza semiprivada	150
23464	204	Televisión	10
23464	245	Rayos X	25
23464	414	Exámenes	35

Imagen 4. Primera forma normal en el ejemplo del Hospital.

Cada grupo repetitivo se deja como una nueva tabla, manteniendo la clave de la cual provienen. Normalmente esta tabla tendrá una clave primaria compuesta por la clave primaria de la tabla original y el atributo del cual dependen los demás atributos del grupo repetitivo.

Segunda Forma Normal (2FN)

Esta forma debe cumplir las siguientes condiciones:

- Debe satisfacer la 1FN.
- Cada atributo debe depender de la clave primaria, y no solo una parte de ella.
- Los atributos que dependen de manera parcial de la clave primaria deben ser eliminados o almacenados en una nueva entidad.

Para pasar una tabla de esta forma, nos debemos fijar en las tablas que tengan claves primarias compuestas, ya que en ellas se pueden dar dependencias parciales. En otras palabras, **si una entidad depende de sólo una parte de la clave primaria compuesta**, se deberá eliminar de esa tabla y llevarla a una nueva con la clave primaria que le corresponde.

Aplicando la Segunda Forma Normal en nuestro ejemplo

En la tabla ***FacturaItem*** tenemos dos claves, el número de factura y el número de ítem.

```
FacturaItem (
  #numero_factura,
  #numero_item,
  nombre_item,
  valor_item
)
```

Cuando ocurre algo de esta naturaleza, hay que determinar si los otros atributos dependen de una clave, o bien de ambas claves.

- nombre_item depende del numero_item.
- valor_item: Pueden ocurrir dos casos.
    - Depende de la relación numero_factura y numero_item, ya que es el valor obtenido para la factura, no unitario.
    - Es el valor unitario y depende de numero_item.

Para solucionar este conflicto, deberemos hablar con nuestro cliente para definir cuál de las dos opciones debemos seguir. Para efectos de este ejercicio, optamos por la segunda opción.

Como tenemos atributos que dependen de ambas claves y otros de una sola clave, tenemos que separarlas en 2 tablas nuevamente.

```
FacturaItem (
  #numero_factura,
  numero_item
)
```

```
Item(
  #numero_item,
  nombre_item,
  valor_item
)
```

Resultando las tablas que verás en la siguiente imagen:

#Factura	ITEM
23464	200
23464	204
23464	245
23464	414

ITEM	NOMBRE	VALOR
200	Pieza semiprivada	150
204	Televisión	10
245	Rayos X	25
414	Exámenes	35

Imagen 5. Segunda Forma Normal para las tablas Item y FacturaItem.

Tenemos la tabla factura que cumple la primera forma normal, por lo que tenemos la siguiente anotación:

```
Factura(
  #numero_factura,
  fecha_factura,
  nombre_paciente,
  #numero_paciente,
  direccion,
  comuna,
  ciudad,
  codigo_sistema_salud
)
```

Acá nuevamente tenemos una clave primaria compuesta del número de la factura y número del paciente. Por lo que debemos determinar cada uno de los atributos si dependen de una clave, o bien de ambas claves.

- `fecha_factura` depende de la relación entre el `numero_factura` y el numero_paciente.
- `nombre_paciente` depende del paciente, en este caso `numero_paciente`.
- `dirección` depende del paciente, en este caso `numero_paciente`.
- `comuna` depende del paciente, en este caso `numero_paciente`.
- `ciudad` depende del paciente, en este caso `numero_paciente`.
- `codigo_sistema_salud` depende del paciente, en este caso `numero_paciente`.

Esto entonces resulta en que tendrémos 2 tablas:

```
Factura(
  #numero_factura,
  numero_paciente,
  fecha_factura
)
```

```
Paciente(
  #numero_paciente,
  nombre_paciente,
  dirección,
  comuna,
  ciudad,
  código_sistema_salud
)
```

Las cuales podemos ver a continuación:

Paciente

#Paciente	Nombre	#Dirección	Comuna	Ciudad	CSS
4724	Marta Fuentes	Monjitas 245	Santiago	Santiago	10

Factura

#Factura	Fecha factura	#Paciente
23464	20/04/19	4724

Imagen 6. Resultado de la 2FN para las tablas Paciente y Factura.

Tercera Forma Normal (3FN)

Esta forma debe cumplir las siguientes condiciones:

- Debe satisfacer 2FN.
- Toda entidad debe depender directamente de la clave primaria.

Si observamos en la imagen, las tablas que tenemos hasta el momento nos preguntamos ¿El paciente tiene alguna relación directa con la comuna y la ciudad en sí?.

Paciente

#Paciente	Nombre	Dirección	id_comuna	CSS
4724	Marta Fuentes	Monjitas 245	145	10

Comuna

id_comuna	Nombre	id_ciudad
145	Santiago	12

Ciudad

id_ciudad	Ciudad
12	Santiago

Imagen 7. Resultado de la 2FN.

La respuesta es no. A esta dependencia se le denomina dependencia funcional transitiva, y se define como la dependencia que tiene un atributo con otro de la misma entidad de forma indirecta y transitiva, es decir, en una tabla de atributos A, B y C, el atributo A depende directamente de B, y a su vez B depende directamente de C.

¿Qué sucede? Que A depende indirectamente y transitivamente de C pues su dependencia directa (B) está dependiendo directamente de C. La dependencia funcional transitiva solo puede ocurrir en entidades con 3 o más atributos en esta fase, si aún quedan atributos que no dependen directamente de la clave primaria, queda llevarlas a una nueva tabla y se puede relacionar a través de una clave foránea con la tabla proveniente. Para pasar a esta forma, debemos encargarnos de eliminar toda dependencia funcional transitiva.

Aplicando la Tercera Forma Normal en nuestro ejemplo

Tenemos la tabla Paciente, en la cual nombre_paciente, numero_paciente, direccion, `código_sistema_salud` dependenden directamente del paciente, pero comuna y ciudad no dependen directamente del paciente, por lo que tenemos que separar estos atributos de esta tabla.

Nuestras nuevas tablas quedarían de la siguiente manera:

```
Paciente(
  #numero_paciente,
  nombre_paciente,
  direccion,
  codigo_sistema_salud,
  id_comuna
)
```

```
Comuna(
  #id_comuna,
  nombre_comuna,
  ciudad
)
```

En este caso, dejamos una `id_comuna` como clave foránea en la tabla Paciente, para hacer referencia a ella.

Ahora, si observamos la tabla resultante Comuna, la ciudad no depende directamente de la `id_comuna`, por lo que nuevamente debemos separar dichos atributos, resultando las siguientes tablas:

```
Comuna (
  #id_comuna,
  nombre comuna,
  id_ciudad
)
```

```
Ciudad(
  #id_ciudad,
  nombre_ciudad
)
```

Nuevamente, `id_ciudad` en la tabla comuna queda como clave foránea a la Tabla ciudad con el id_ciudad.

Para el resultado de este ejercicio, nos quedan las siguientes tablas en 3FN expuestas en la imagen:

Paciente

#Paciente	Nombre	Dirección	id_comuna	CSS
4724	Marta Fuentes	Monjitas 245	145	10

Comuna

id_comuna	Nombre	id_ciudad
145	Santiago	12

Ciudad

id_ciudad	Ciudad
12	Santiago

Imagen 8. Tercera Forma Normal.

En un modelo de bases de datos quedaría representado de la forma expuesta en la siguiente imagen.

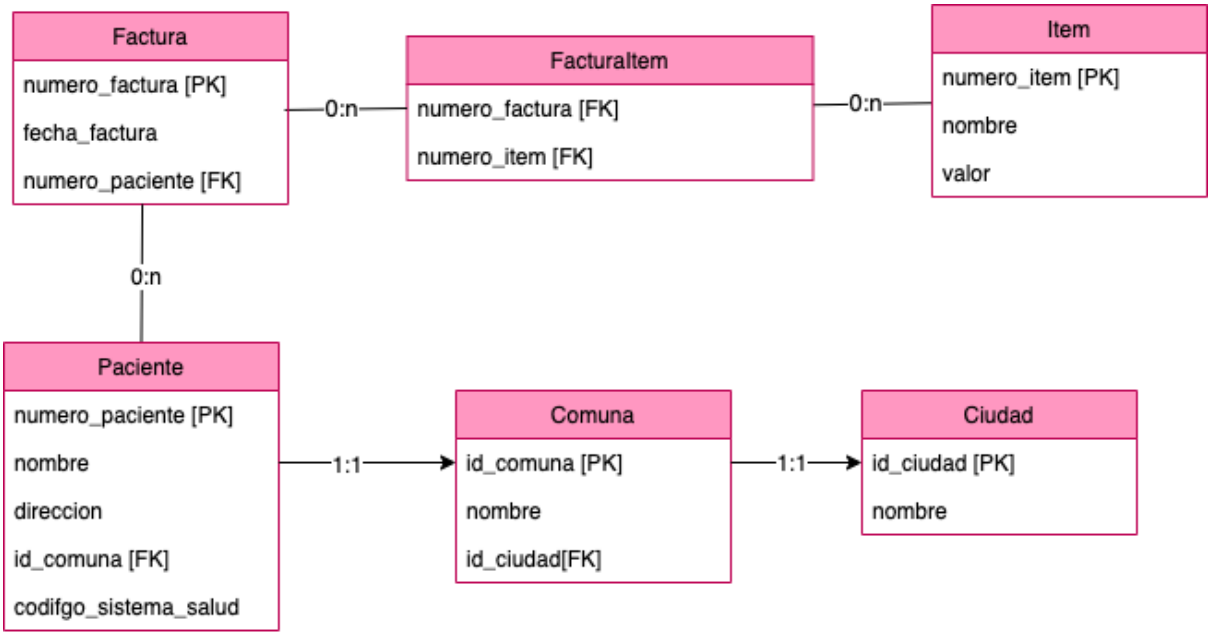

Imagen 9. Modelo de base de datos.

Resumen de la normalización

Como podemos ver, el proceso de normalización conlleva una serie de pasos ordenados que nos permiten reducir la cantidad de datos que debemos almacenar en una base de datos, evitando así la redundancia de datos, simplifica la actualización de los datos y garantiza integridad referencial, previniendo la inconsistencia en las relaciones.

Como podemos ver en la siguiente imagen, en cada paso de la normalización se va reduciendo la cantidad de datos que tenemos en nuestro universo de la base de datos.

Imagen 10. Resumen de Normalización.

¿Qué pasaría si no hubiésemos normalizado nuestra base de datos?

Si pasamos nuestra vista inicial de factura a una tabla, sin normalizar, tenemos la tabla de la siguiente imagen donde solo estamos mostrando los registros correspondientes a Marta Fuentes.

numero factura	fecha factura	nombre paciente	numero paciente	direccion paciente	comuna paciente	ciudad paciente	Cod-sis-salud	numero item	nombre item	valor item
23464	20/04/2019	Marta Fuentes	4724	Monjitas 245	Santiago	Santiago	10	200	Pieza semiprivada	150
23464	20/04/2019	Marta Fuentes	4724	Monjitas 245	Santiago	Santiago	10	204	Televisión	10
23464	20/04/2019	Marta Fuentes	4724	Monjitas 245	Santiago	Santiago	10	245	Rayos X	25
23464	20/04/2019	Marta Fuentes	4724	Monjitas 245	Santiago	Santiago	10	414	Exámenes	35

Imagen 11. Tabla sin normalizar.

Imaginemos que queremos cambiar la dirección de nuestro paciente. Al tener todo desnormalizado, antes hubiésemos tenido que ir a buscar en cada uno de los registros de

las facturas y cambiar la dirección del paciente correspondiente. En cambio, al tener normalizado, basta que consultemos la tabla de Pacientes y realizar la actualización en dicha tabla.

Lo bueno de tener la base de datos completamente normalizada es que minimizamos el espacio necesario para almacenar los datos y reducimos las anomalías de mantención.

Desnormalización y sus usos

Por desnormalización, entendemos el proceso de añadir redundancia en las tablas de manera deliberada. Una de las motivaciones para desnormalizar tablas en una base de datos es el reducir el tiempo de consulta al implementar joins en múltiples tablas.

Un elemento importante a considerar es el hecho que desnormalizar no significa ignorar el proceso de normalización. Más bien, es un paso posterior a la normalización donde buscamos maximizar la eficiencia y representación de los datos, a expensas de hacer más compleja la mantención de una tabla específica.

En una base de datos tradicional buscamos modularizar las entidades asociadas a un fenómeno en distintas tablas para reducir la redundancia y problemas lógicos en éstas. En una base de datos desnormalizada, preferimos aumentar la redundancia de nuestra base para reducir la cantidad de consultas mediante joins y ganar eficiencia en las consultas.

Ejemplo de desnormalización

Tenemos una base de datos sobre pacientes en un hospital, la cual está compuesta por tres tablas.

- Una tabla Paciente, que registra el nombre del paciente, el tipo de tratamiento y el doctor con el que se atendió, éstas últimas dos columnas están registradas como claves foráneas así como ves en la siguiente tabla.

Paciente	Tratamiento_id	Doctor_id
María Zenteno	1	1
Fernando Sánchez	2	2
Jose Artigas	1	2
Emilia Tapia	1	1
Felipe Zamorano	2	1
César Oyarce	2	1
Catalina Maillard	1	2

Imagen 12. Ejemplo 1 de desnormalización.

- Una tabla Doctores que vincula la clave primaria con el nombre del doctor tratante como en la siguiente tabla.

Id	Nombre
1	Astorquiza
2	Feris

Imagen 13. Ejemplo 2 de desnormalización.

- Una tabla Tratamiento que vincula la clave primaria con el nombre del tratamiento asignado al paciente.

Id	Nombre
1	Astorquiza
2	Feris

Imagen 14. Ejemplo 3 de desnormalización.

Resulta que por sí sola, la tabla Paciente no nos informa con quién se trataron ni qué tratamiento tuvieron. Para generar sentido sobre estos números, es necesario complementar la información existente en la tabla Paciente con las tablas Doctores y Tratamiento.

Se observa que la tabla Doctores presenta una cantidad menor de datos registrados, lo cual nos permite hacer uso eficiente de la memoria en nuestra base de datos. Lo mismo se aplica para el caso de la tabla Tratamientos.

Para este ejemplo práctico, observamos que la aplicación de consultas mediante el comando JOIN no es lo suficientemente compleja, dada la baja cantidad de casos y la simpleza de las consultas utilizadas.

Una tabla desnormalizada implica duplicar la información de manera deliberada en una nueva representación de todas las tablas solicitadas.

Verás ahora la tabla desnormalizada que permitirá asociar cada paciente con un doctor y un tratamiento específico.

Paciente	Tratamiento_id	Tratamiento_non	Doctor_id	Doctor_nombre
María Zenteno	1	Control Médico	1	Astorquiza
Fernando Sánchez	2	Biopsia	2	Feris
Jose Artigas	1	Control Médico	2	Feris
Emilia Tapia	1	Control Médico	1	Astorquiza
Felipe Zamorano	2	Biopsia	1	Astorquiza
César Oyarce	2	Biopsia	1	Astorquiza
Catalina Maillard	1	Control Médico	2	Feris

Imagen 15. Tabla desnormalizada.

¿Y por qué debemos desnormalizar?

La desnormalización busca como objetivo optimizar el funcionamiento de una base de datos con el costo de agregar datos redundantes. El precio de agregar información adicional en una tabla puede generar que a la larga cueste menos con respecto a las consultas que se deba estar haciendo de manera reiterada, en pocas palabras es un dilema entre almacenar menos registros pero acomplejar las consultas a la base de datos versus ofrecer un tiempo de respuesta optimo a cambio de tener redundancia en las tablas. Esta no es una decisión genérica pues siempre dependerá del caso de uso y la problemática en evaluación.