Modelación y Normalización de datos

Modelación

Hasta el momento, nuestro conocimiento sobre bases de datos es estrictamente procedimental: sabemos cómo implementar tablas con columnas en un motor de bases de datos, así como generar instrucciones de modificación de tablas. El siguiente paso es saber modelar los problemas, o sea construir el modelo de datos e implementarlo en base a un problema.

La modelación de bases de datos es un proceso para definir e implementar requerimientos del usuario/empresa dentro del contexto de los sistemas de información disponibles en una organización, o sea partir de uno o mas requerimientos construiremos una base de datos que nos permita almacenar y recuperar información relevante.

Elementos básicos de una base de datos

Para entender el proceso de modelado de datos, debemos definir los elementos básicos de una base de datos.

La unidad básica en una base de datos se conoce como **entidad**, que corresponde a un objeto del mundo real. De esta entidad provienen los atributos, que representan a características únicas de la entidad. Por ejemplo, el señor Juan representaría una entidad, la cual tiene asociada una serie de atributos tales como Nombre, Apellido y Edad. Del ejemplo anterior surge la pregunta ¿Qué puede ser considerado como una entidad o atributo? Resulta que la definición dependerá de los requerimientos y casos de uso.

Por ejemplo, si estamos realizando una base de datos de clientes para una automotriz, probablemente queramos considerar a cada cliente como entidad con atributos como auto comprado, año del auto, etc. De similar manera, si en la misma automotriz nos piden generar una base de datos sobre automóviles disponibles para la venta, nuestra entidad (o unidad de medición) cambiará, así como sus atributos reflejarán características del auto.

Los ejemplos mencionados buscan explicitar la importancia de la modelación: antes de comenzar a ingresar y preprocesar nuestros datos para futuras aplicaciones, debemos modelar la naturaleza del problema y como las entidades se relacionan entre sí. El modelo es el paso previo a la construcción. Es importante que el modelamiento sea el correcto, dado que puede afectar la fidelidad de los datos y complicar el desarrollo de software que lo requiera. Para efectos prácticos de esta lectura, trabajaremos con bases de datos relacionales (bases de datos representadas en tablas).

¿Cómo hacemos un modelo de datos?

El proceso de modelo de datos es una serie de 3 pasos que permiten bajar la abstracción del requerimiento del cliente a lo concreto. Estos pasos son:

- Modelo Conceptual.
- Modelo Lógico.
- Modelo Físico.

El modelo conceptual

El primer paso es el modelo conceptual, en el cual tomamos los requerimientos del cliente y construimos un diagrama que represente las entidades mediante tablas, y sus atributos mediante campos de cada tabla.

Estos diagramas pueden ser considerados innecesarios por personas con poca experiencia en el área de desarrollo, pero los modelos, ya sean conceptuales, lógicos y/o físicos son muy útiles para comunicar distintas necesidades y coordinar esfuerzos a la hora de trabajo. Es normal verlos dibujados en pizarras o impresos y colgados en las paredes de las oficinas de trabajo.

Aprender a crear modelos conceptuales nos entregará herramientas importante para modelar como:

- Identificar entidades y atributos.
- Identificar relaciones y sus cardinalidades.
- Proponer identificadores.

Para construir el modelo conceptual es vital identificar las unidades de medición con las que trabajaremos, cómo serán los registros ingresados en la base y cómo consideraremos la conceptualización de los atributos. La principal característica de este modelo es que debe ser entendible por todos los clientes, independiente de que sean programadores.

Ejercicio 1

Una empresa vende productos a varios clientes. Se necesita conocer los datos personales de los clientes (nombre, apellido, dni y dirección).

El primer paso es identificar las entidades, en este caso hay 3 candidatos interesantes, empresa, productos y clientes, el resto nombre, apellido, dni y dirección son atributos de los clientes.

De las entidades no nos piden llevar registro de las empresas, hay una sola y no hay requisitos de información respecto a ésta, así que la descartamos, de los productos no nos piden nada en específico por lo que podríamos desecharla.

Para construir el modelo conceptual dibujaremos cada entidad como un cuadrado y cada atributo como un círculo (elipsis):

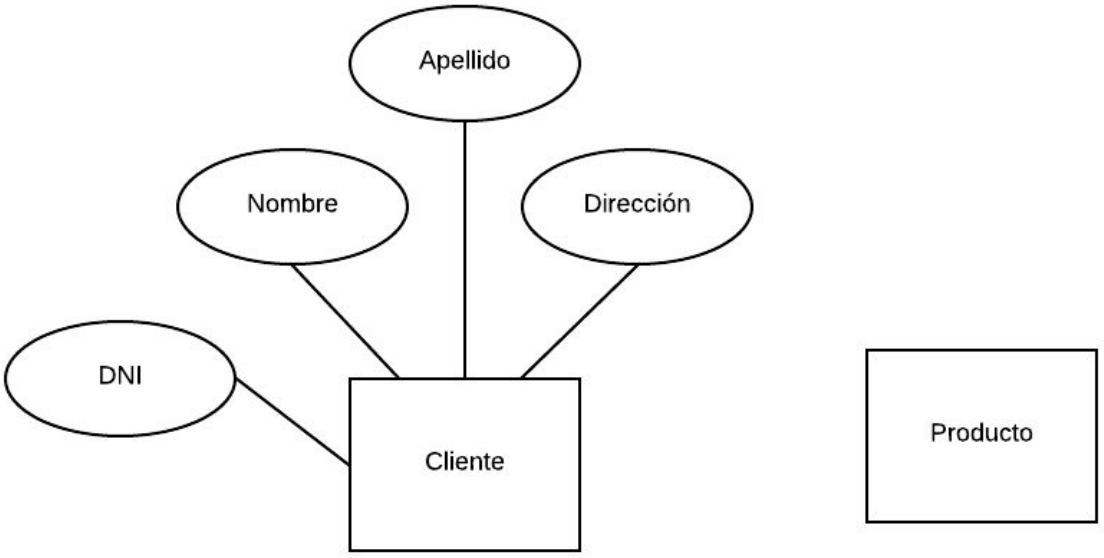

Imagen 1. Modelo conceptual ejercicio 1.

Al mostrarle nuestro diagrama a nuestro cliente el nos dirá que está muy bien pero que no refleja algo que nos dijo y que no anotamos en una primera etapa, que un cliente puede comprar varios productos y un producto puede ser comprado por varios clientes y nos pide que actualicemos nuestros diagrama para reflejar esto.

Además nuestro cliente aprovecha la oportunidad de decirnos que los productos tienen un código.

Entonces tenemos que aprender a agregar relaciones en un modelo conceptual, para indicar la relación utilizaremos rombos.

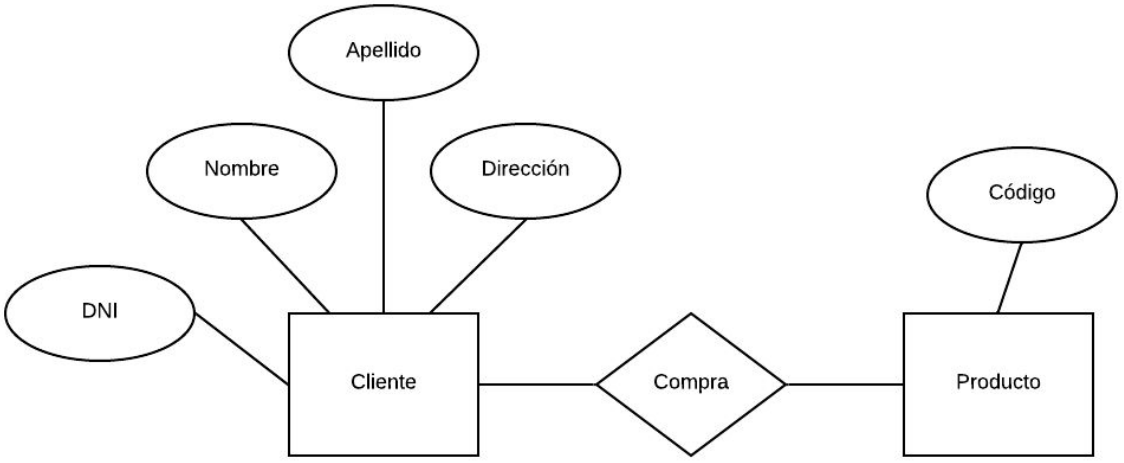

Imagen 2. Agregando relaciones.

Ejercicio 2:

Un cliente necesita construir una plataforma para llevar registro de los usuarios que se inscriben en un formulario de una página web, en el formulario llenan email, nombre y teléfono. Por otro lado el sistema tiene que ser capaz de llevar registro de cada llamado telefónico que se la ha hecho al cliente y un reporte asociado a ese llamado y que persona realizó el llamado.

Paso 1: Detección de entidades

- **Entidades:** Usuario y reporte.

Existe otro candidato a entidad, quien realiza la llamada, sin embargo no hay información de sus posibles atributos, por ahora no lo agregaremos.

Paso 2: Agrupamos entidades con sus atributos

```
Usuario (email, nombre, teléfono)
Registro (quien llamó, reporte, fecha y hora)
```

Paso 3: Nombrar la relación, en este caso la relación sucede de una llamada telefónica y la podemos llamar así:

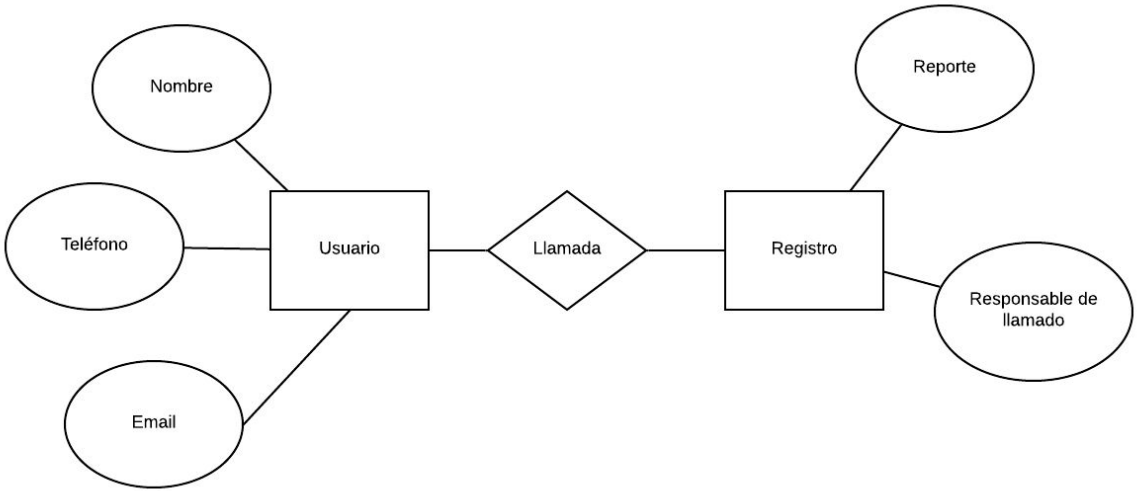

Imagen 3. Relación de llamada telefónica.

Ejercicio 3:

Un banco que calcula riesgo de clientes necesita llevar un registro de personas con sus nombres y números telefónicos, el país donde nacieron, sus vehículos con sus patentes, marca, modelo y año y sus propiedades con su dirección, metros cuadrados y metros cuadrados construidos. Nos piden hacer el modelo conceptual.

Finalmente cabe mencionar que existen otros tipos de diagramas para realizar modelos conceptuales, el que estamos ocupando es uno de los mas famosos fue diseñado por Peter Chen en 1976.

Resumiendo

El modelo conceptual nos ayuda a describir y comunicar el contenido de nuestra base de datos a un nivel alto, o sea describirlo de forma independiente de las estructuras donde se guardarán nuestros datos. Para desarrollarlo necesitamos seguir estos 3 pasos principales:

- Definir las entidades.
- Agrupar las entidades con sus atributos.
- Definir las relaciones entre las entidades.

Cardinalidad

La cardinalidad indica el número mínimo y máximo de entidades con las que puede relacionarse una entidad, se indica como **mínimo:máximo**. Ésta puede ser cualquier combinación de valores arbitrarios. Hay 3 combinaciones claves que definen la naturaleza de la relación.

1:1

Donde cada registro **debe** estar asociado a otro registro:

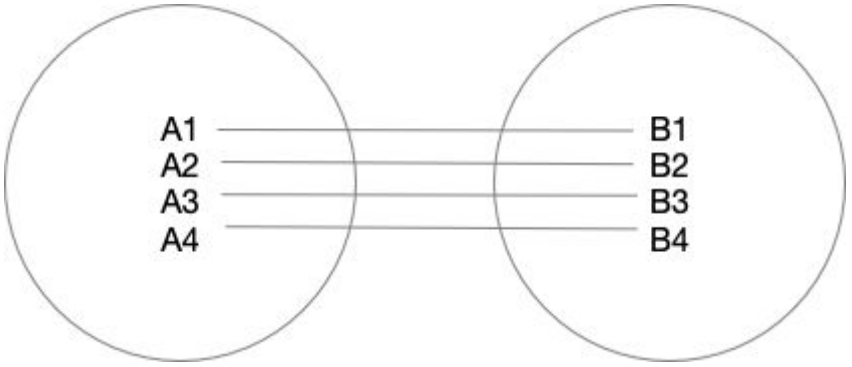

Imagen 4. Cardinalidad 1:1.

Otras variantes del mismo tipo, a veces incluso de uso mucho más frecuente, son las cardinalidades de 0:1 donde uno de los registros **puede** estar asociado a otro registro.

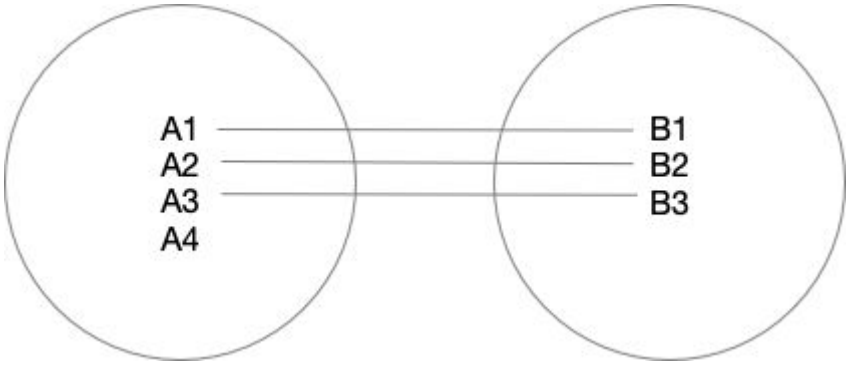

Imagen 5. Cardinalidad 0:1.

Un ejemplo de esta cardinalidad sería un modelo con personas y sus cónyuges. Una persona **puede** tener una relación con otra, entonces esta sería una cardinalidad de 0:1.

1:N

Donde cada entidad puede estar asociada a cualquier número de otras entidades:

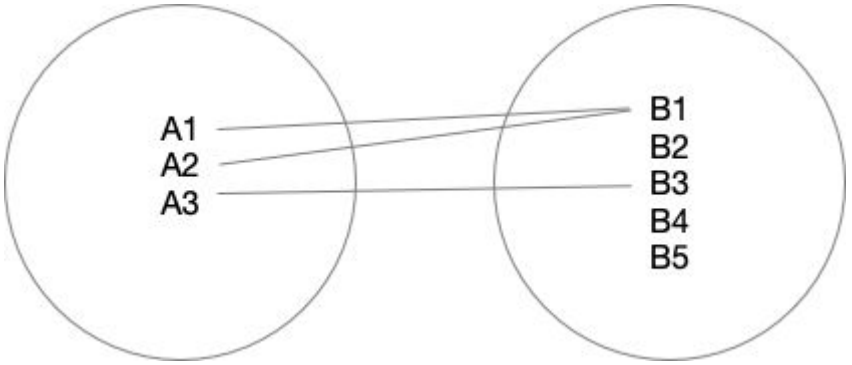

Imagen 6. Cardinalidad 1:N.

Este es el tipo de cardinalidad más frecuentemente utilizada. Un ejemplo común de esta cardinalidad es el uso de **categoría**.

Por ejemplo, tenemos artículos y cada uno puede tener una categoría; por lo mismo en categoría hay diversos artículos.

Otro ejemplo similar son personas y sus países de orígenes, una persona solo puede tener un país de origen, cada país tiene asociado N personas.

Otras variantes típicas son 0:N y fijando un N por ejemplo 1:4.

N:N

Este tipo de cardinalidad representa situaciones donde cada entidad puede estar asociadas a múltiples entidades de otro tipo y al mismo tiempo una entidad del otro grupo:

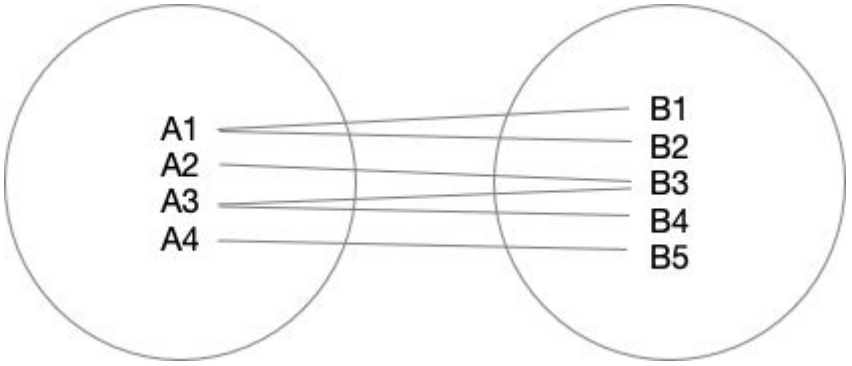

Imagen 7. Cardinalidad N:N.

Un ejemplo típico de esta cardinalidad se da entre artículos y tags. Un artículo puede tener múltiples tag y cada tag puede estar asociado a múltiples artículos.

Un detalle importante como ambas cardinalidades pueden ser distintas a este tipo de cardinalidad a veces se le dice N:M.

Ejemplos de cardinalidad

Si decimos que la cardinalidad de una relación entre clientes y cuentas bancarias es de 1 a N, esto quiere decir que un cliente puede tener varias cuentas bancarias y que una cuenta bancaria solo puede pertenecer a un cliente. Si decimos que es de 1 a 1, estamos diciendo que un cliente solo puede tener una cuenta bancaria y esa cuenta bancaria le debe pertenecer a un único cliente.

Por ejemplo, si volvemos a nuestro ejercicio donde guardabamos el registro de llamadas telefónicas de diversos clientes, veremos que a cada cliente podemos hacer varios llamadas que resulten en registros, y un registro de llamada telefónica sólo puede pertenecer a un cliente.

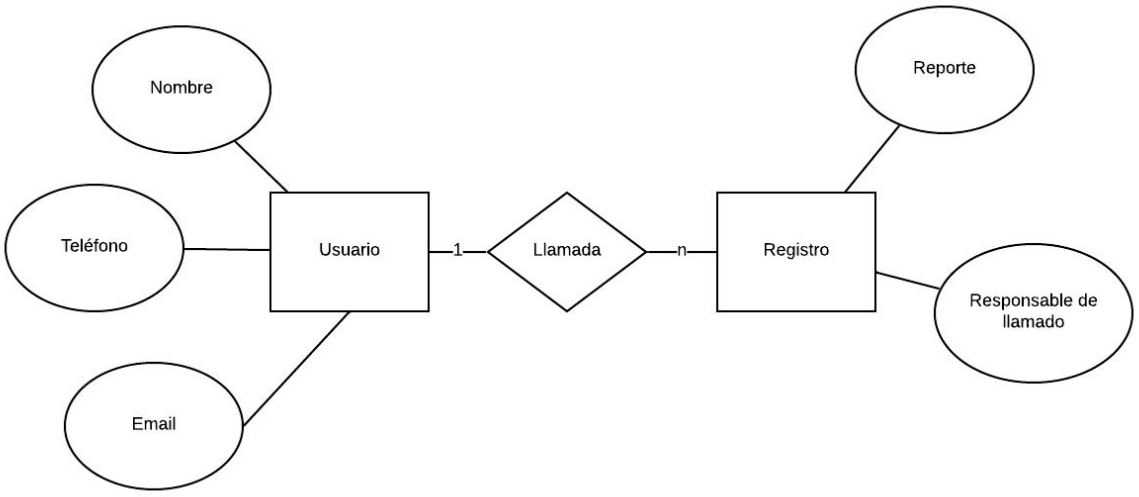

Imagen 8. Relaciones con cardinalidad.

Identificadores

El identificador es un atributo o conjunto de atributos que determina de modo único la ocurrencia de una entidad. Veamos algunos casos:

Tenemos un registro de trabajadores de una oficina, de los cuales tenemos nombre, email corporativo y fecha de nacimiento ¿Cuál debería ser el identificador?

El nombre y la fecha de nacimiento pueden repetirse, por lo que email corporativo es el mejor candidato. ¿Qué pasaría si por algún motivo dos personas ocupan el mismo email, por ejemplo: `finanzas@corporación.com`? En ese caso, otra posibilidad sería utilizar un atributo compuesto, por ejemplo: mail más nombre, pero en una empresa grande podría darse que eso tampoco sea único. Podríamos agregar la fecha de nacimiento o incluso podríamos crear un atributo nuevo que sea un código trabajador para identificar únicamente a cada trabajador.

Suele suceder que en el proceso de modelamiento aparecen estas necesidades para las empresas y proyectos y por eso se debe diseñar esta etapa con cuidado. Eventualmente los identificadores serán nuestras claves primarias.

Por ahora diremos que los email corporativos son únicos y cada persona en la empresa tiene el suyo propio.

En el caso de los registros no tenemos un buen candidato.

El reporte es un texto largo que no tenemos seguridad de que no sean iguales entre ellos. Resulta que una persona puede llamar múltiples veces. La combinación de estos tampoco hay seguridad de que sea única. Tenemos dos opciones interesantes: agregar la hora de la llamada o agregar un número para esta llamada. **Debemos consultar con nuestros clientes antes de agregar entidades y atributos.**

Por el bien del ejercicio, supongamos que nuestro cliente nos dio el visto bueno para agregar un identificador a la tabla de registros de llamados. De esta forma, subrayando los identificadores nuestro modelo conceptual quedaría de la siguiente forma:

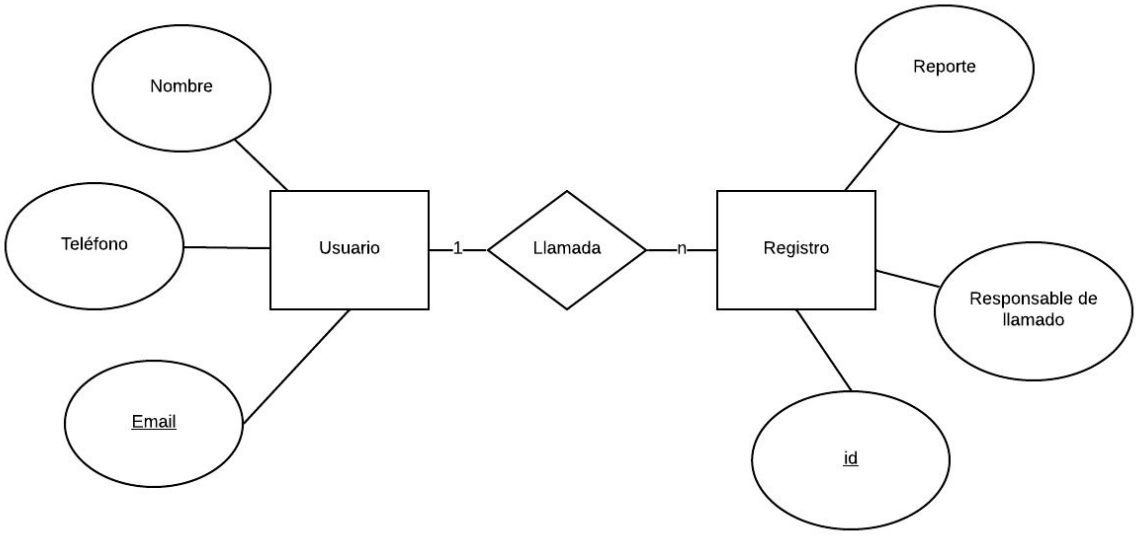

Imagen 9. Modelo conceptual actualizado.

¿Por qué son importantes los identificadores?

El concepto más importante asociado a bases de datos es el de **integridad**: que nuestros registros sean correctos y completos. Sin identificadores no tendríamos certeza que estamos modificando el dato correcto. Por ejemplo, si cambiamos el teléfono de un usuario con nombre Juan, pero hay cincuenta usuarios con ese nombre, cambiaríamos los 50 teléfonos perdiendo nuestros datos anteriores.

Los identificadores nos permiten a encontrar de forma única nuestros datos. Sobre estos construiremos nuestras claves primarias más adelante en el modelado lógico.

Un detalle importante: es perfectamente posible tener una tabla sin identificadores, pero esto puede ser problemático por los motivos anteriormente expuestos. Un buen modelado conceptual requiere de definir identificadores para cada entidad.

El modelo lógico

Mientras el modelo conceptual tiene como objetivo describir la información de forma independiente del motor de base de datos, el modelo lógico describe el modelo de datos en términos del tipo de motor. En esta unidad, trabajaremos con un motor relacional.

Para transformar un modelo conceptual en lógico lo que tenemos que hacer es:

1. Transformar todas las entidades en tablas, agregar los atributos como columnas de la tabla.
2. Transformar todas las relaciones del tipo **N:N** en tablas.
3. Propagar la clave primaria de las tablas en las relaciones **1:N** desde el lado de las 1 al lado de las N.

Realicemos el ejemplo con nuestro modelo conceptual:

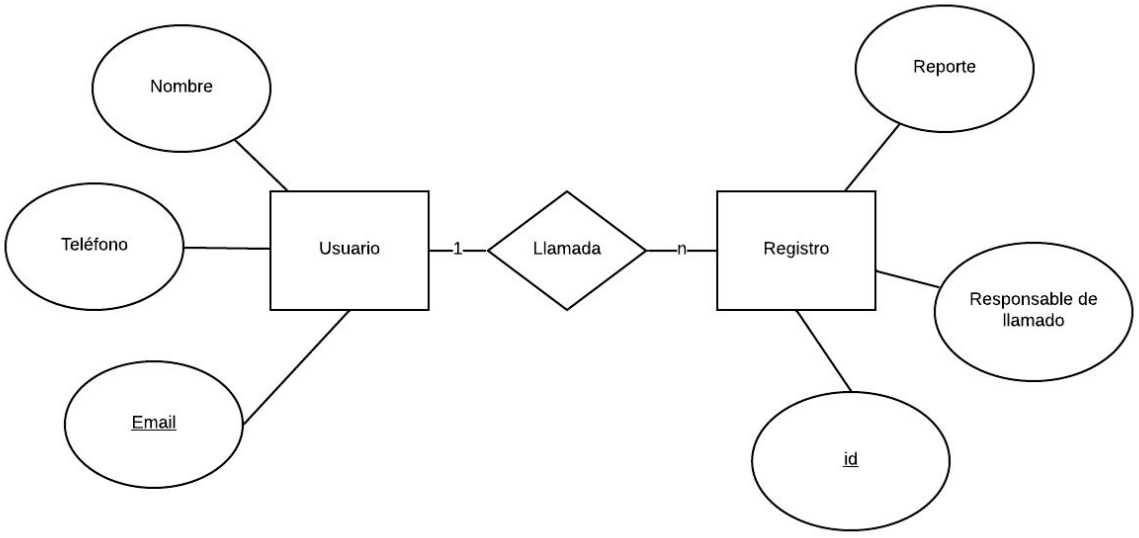

Imagen 10. Modelo conceptual.

1. Tenemos 2 entidades por lo que quedarán dos tablas.
2. No tenemos relaciones N a N así que no se crearán tablas nuevas.
3. Propagamos la clave primaria desde el lado de las 1 a las N.

Para realizar el diagrama del modelo lógico hay distintos tipos de notaciones.

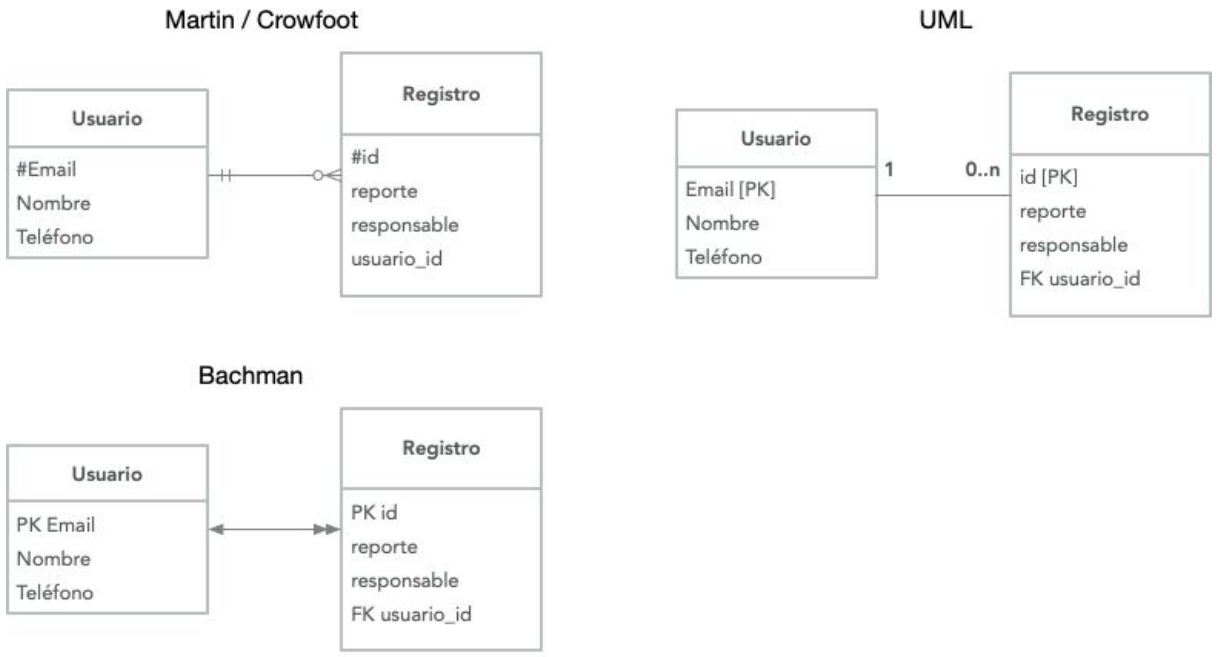

Imagen 11. Notaciones.

La diferencia entre los tipos de notaciones principalmente cambian la forma de representar la cardinalidad y como anotar la clave primaria, foránea y atributos obligatorios.

Existen estándares para representar cada uno de los diagramas, sin embargo, es poco habitual que los programas para realizar estos diagramas se rijan 100% en función de estos estándares. A lo largo de esta unidad, utilizaremos la notación de UML. En esta la cardinalidad de la relación se anota como `mínimo..máximo`, un único número indica qué la cantidad es obligatoria. El nombre de la relación va indicado sobre la línea para indicar la relación de izquierda a derecha y bajo la línea para indicarla de derecha a izquierda.

¿Cómo recordar la propagación de clave?

En lugar de memorizar reglas sobre cuál tabla propaga a cuál, es mejor visualizar cómo sería el resultado.

Supongamos que tenemos la tabla de artículos y la tabla de comentarios. Un artículo puede tener muchos comentarios, y un comentario le pertenece a un artículo ¿En qué lado va la clave foránea?

Artículo

id	url	título	contenido
1	/articulo1	Artículo 1	Lorem Ipsum ...
2	/articulo2	Artículo 2	Lorem Ipsum ...
3	/articulo3	Artículo 3	Lorem Ipsum ...
4	/articulo4	Artículo 4	Lorem Ipsum ...

Comentario

id	contenido
1	Primer comentario
2	Segundo comentario
3	Tercer comentario
4	Cuarto comentario

Imagen 12. Tablas Artículo y Comentario.

Hay solo dos opciones, en el lado del artículo y en el de comentarios. Si ponemos la clave de artículo en los comentarios quedaría:

Artículo

id	url	título	contenido
1	/articulo1	Artículo 1	Lorem Ipsum ...
2	/articulo2	Artículo 2	Lorem Ipsum ...
3	/articulo3	Artículo 3	Lorem Ipsum ...
4	/articulo4	Artículo 4	Lorem Ipsum ...

Comentario

id	contenido	articulo_id
1	Primer comentario	1
2	Segundo comentario	1
3	Tercer comentario	2
4	Cuarto comentario	1

Imagen 13. Mostrando la clave del artículo en los comentarios.

En cambio si lo hiciéramos al revés tendríamos algo así:

Artículo

id	url	título	contenido	comentario1_id	comentario2_id	comentario3_id	comentario4_id
1	/articulo1	Artículo 1	Lorem Ipsum ...	1	2		4
2	/articulo2	Artículo 2	Lorem Ipsum ...			3	
3	/articulo3	Artículo 3	Lorem Ipsum ...				
4	/articulo4	Artículo 4	Lorem Ipsum ...				

Comentario

id	contenido
1	Primer comentario
2	Segundo comentario
3	Tercer comentario
4	Cuarto comentario

Imagen 14. Mostrando la clave de los comentarios en los artículos.

Aquí necesitaríamos infinitas columnas para poder guardar las asociaciones, por lo mismo la única opción viable es agregar la clave foránea en el lado de los N. Si olvidamos la regla siempre podemos hacer este ejercicio de visualización.

El modelo físico

El modelo físico es la versión final del modelo.

En esta etapa se agregan los tipos de datos asociados a cada atributo y se dejan explícitos los índices.

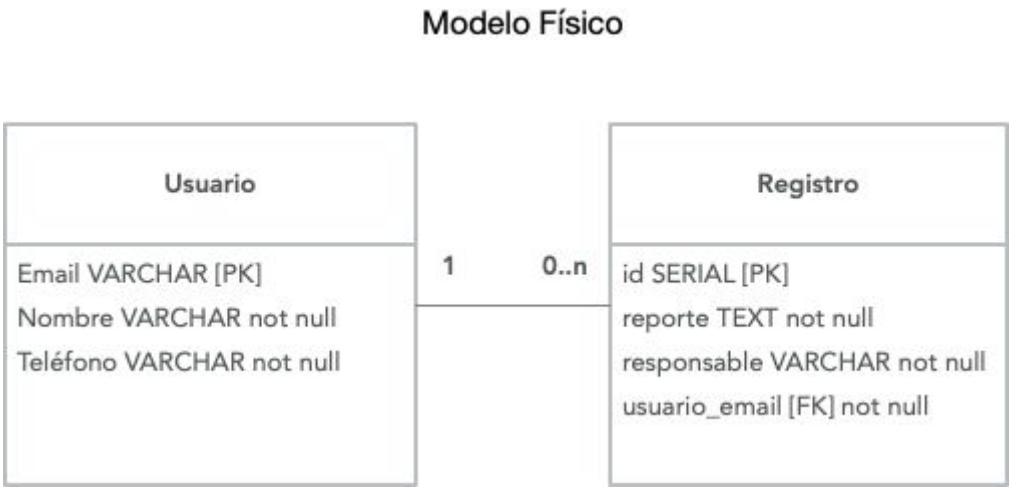

Imagen 15. Modelo físico.

Traspasando el modelo físico a SQL

Con el modelo físico podemos pasar construir nuestras tablas y agregar datos.

```
CREATE TABLE usuario(
  email VARCHAR,
  nombre VARCHAR,
  telefono VARCHAR,
  PRIMARY KEY (email)
);

CREATE TABLE registro(
id SERIAL,
  reporte TEXT,
  responsable VARCHAR,
  usuario_email VARCHAR REFERENCES usuario (email),
  PRIMARY KEY (id)
);
```

Al traspasar los datos a SQL nos damos cuenta que nuestro modelo físico tiene toda la información necesaria.

Existen programas que nos permiten generar SQL a partir de modelos físicos y otros que nos permiten generar los diagramas a partir de SQL.

Finalmente agregamos datos para probar nuestro modelo.

```
INSERT INTO usuario(email, nombre, telefono) VALUES
('usuario1@gmail.com', 'Juan', '12345678');
INSERT INTO usuario(email, nombre, telefono) VALUES
('usuario2@gmail.com', 'Francisca', '12345679');
INSERT INTO registro (reporte, responsable, usuario_email) values ('El
usuario presenta problemas para realizar el pago online', 'Javiera',
'usuario1@gmail.com');
INSERT INTO registro (reporte, responsable, usuario_email) values ('El
usuario presenta problemas para ingresar a la plataforma', 'Javiera',
'usuario2@gmail.com');
```

Luego si queremos seleccionar todos los registros con sus usuarios realizaremos un join entre ambas tablas.

```
select * from registro join usuario on usuario.email =
registro.usuario_email;
```

Caso con relaciones N a N

Un caso muy común al momento de modelar consiste el de relaciones N a N. Estudiemos el siguiente caso:

Un empleado puede trabajar en diversos proyectos de una empresa, los empleados para entrar a la plataforma necesitan identificarse con un correo corporativo, password y su nombre, de cada proyecto se tiene el nombre y una descripción.

Se pueden dar situaciones donde en un proyecto no trabaje ningún empleado y un empleado no trabaje en ningún proyecto.

Modelo conceptual

Recordemos nuestros pasos importantes:

1. Identificar entidades, en este caso empleado y proyecto.

2. Agrupar entidades con sus atributos..

```
Empleado (email, password, nombre)
Proyecto (nombre, descripción)
```

3. Identificamos relaciones y sus cardinalidades.

Un empleado puede trabajar en 0 a n proyectos, en un proyecto. La relación es "participación"

4. Identificamos candidatos únicos.

En el caso de usuario el email es único, en el caso de proyecto el nombre podría serlo pero eventualmente dos proyectos podrían quedar con el mismo nombre, agregaremos el atributo id.

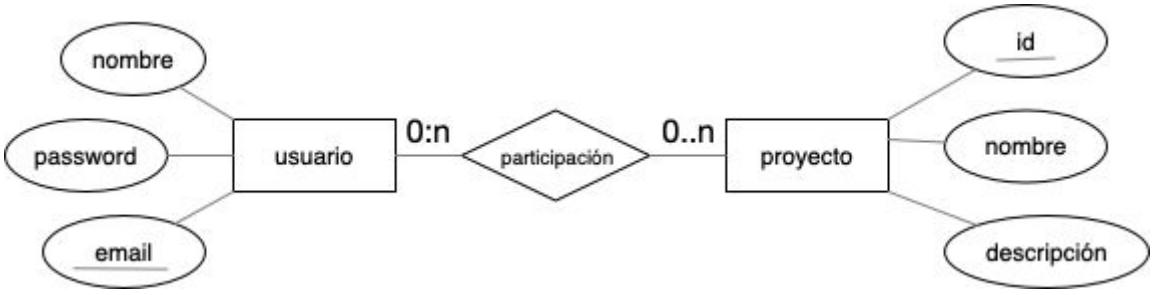

Imagen 16. Modelo conceptual Usuario vs Proyecto.

Para traspasar el modelo a lógico recordemos nuestros pasos:

1. Transformar todas las entidades en tablas, agregar los atributos como columnas de la tabla.
2. Transformar todas las relaciones del tipo N:N en tablas
3. Propagar la clave primaria de las tablas en las relaciones 1:N desde el lado de las 1 al lado de las N.

En este caso tenemos dos entidades más la entidad de la relación N a N.

Al propagar las claves quedaremos con el siguiente diagrama.

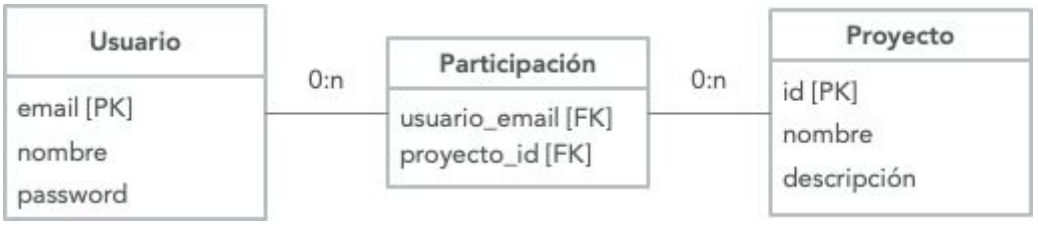

Imagen 17. Modelo lógico Usuario vs Proyecto.

Pero todavía nos queda hacernos unas preguntas importantes:

- ¿Puede un usuario trabajar dos veces en el mismo proyecto?
- ¿Cómo evitamos que esto suceda?

Para evitar los grupos repetitivos o sea que el usuario con email 123@123 trabaje en el proyecto 1 varias veces, podemos agregar una clave primaria compuesta. De esta forma nos aseguramos que los trabajos siempre tengan ambos campos y además que estos existan asegurando la integridad referencial.

```
CREATE TABLE usuario(
  email VARCHAR,
  nombre VARCHAR,
  password VARCHAR,
  PRIMARY KEY (email)
);

CREATE TABLE proyecto(
id SERIAL,
  nombre VARCHAR,
  descripcion text,
  PRIMARY KEY (id)
);

-- La tabla intermedia se crea al último para poder agregar las
referencias

CREATE TABLE participacion(
usuario_email VARCHAR REFERENCES usuario (email),
  proyecto_id INTEGER REFERENCES proyecto (id),
  PRIMARY KEY(usuario_email, proyecto_id)
);
```

Insertamos datos para probar nuestra consulta

```
INSERT INTO usuario(email, nombre, password) VALUES
('usuario1@gmail.com', 'Juan', '12345678');
INSERT INTO usuario(email, nombre, password) VALUES
('usuario2@gmail.com', 'Francisca', 'asdfghi');
INSERT INTO proyecto(nombre, descripcion) VALUES ('Proyecto1', 'Proyecto
secreto');
INSERT INTO proyecto(nombre, descripcion) VALUES ('Proyecto2', 'Proyecto
público');
INSERT INTO participacion(usuario_email, proyecto_id) values
('usuario1@gmail.com', 1);
INSERT INTO participacion(usuario_email, proyecto_id) values
('usuario1@gmail.com', 2);
INSERT INTO participacion(usuario_email, proyecto_id) values
('usuario2@gmail.com', 1);
```

Para seleccionar todos los usuarios con sus proyectos respectivos necesitamos hacer dos joins, usuarios con participación y participación con proyecto.

```
SELECT * FROM usuario
INNER JOIN participacion ON email = usuario_email
INNER JOIN proyecto ON proyecto.id = participacion.proyecto_id;
```

email	nombre	password	usuario_email	proyecto_id	id	nombre	descripcion
usuario1@gmail.com	Juan	12345678	usuario1@gmail.com	1	1	Proyecto 1	Proyecto secreto
usuario1@gmail.com	Juan	12345678	usuario1@gmail.com	2	2	Proyecto 2	Proyecto público
usuario2@gmail.com	Francisca	asdfghi	usuario2@gmail.com	1	1	Proyecto 1	Proyecto secreto

Tabla 1. JOIN usuarios con partición y participación con proyecto.

Normalización

La normalización es un proceso que elimina la redundancia de una base de datos.

El proceso busca evaluar una base de datos a lo largo de una serie de etapas. En cada una de éstas se van eliminando distintos tipos de redundancias. Estas etapas son muy importantes y reciben el nombre de **formas normales**. Nuestro propósito en este capítulo es aprender a llevar una base de datos a tercera forma normal.

¿Para qué sirve la normalización?

La normalización es como una fórmula que podemos aplicar a una base de datos, a un modelo lógico o a incluso a una tabla de excel o 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. Sin embargo en cada etapa lo que estamos haciendo es normalizar.

Para pasar una base de datos de un etapa a la otra tenemos que aplicar un conjunto de reglas.

Notación

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

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

Fuera de los paréntesis tenemos el nombre de la tabla con la que trabajaremos. Dentro de los paréntesis se encuentran los atributos existentes. Hay casos en que dentro de una tabla pueden existir **atributos repetidos**. Para denotarlos de forma explícita, los encapsulamos entre llaves {} de la siguiente forma:

```
Tabla(atributo_1, atributo_2, ..., atributo_n, {atributo_repetido_1,
..., atributo_repetido_j})
```

¿Qué es un atributo repetido?

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

Nº Cliente	Nombre	Teléfonos
1	Fernanda	123-456 123-457
2	Felipe	234-412
3	Francisco	123-411

Tabla 2. Ejemplo de tabla.

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:

Nº Cliente	Nombre	Teléfono1	Teléfono2
1	Fernanda	123-456	123-457
2	Felipe	234-412	
3	Francisco	123-411	

Tabla 3. Ejemplo 2 de tabla.

En ambos casos los atributos se repiten. Ahora si solo existiera la opción de dos 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:

```
Tabla(Nº Cliente, Nombre, {Teléfono})
```

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.

Cuando estemos frente a **claves primarias compuestas**, todos los atributos correspondientes a claves serán encapsulados entre negritas.

- Ejemplo de clave primaria en una tabla:

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

- Ejemplo de clave primaria y foránea en una tabla:

```
Tabla2(#clavePrimaria, claveForánea, ..., atributo_n)
```

- Ejemplo de clave primaria compuesta:

```
Tabla3(#clavePrimaria, #clavePrimariayForánea, ..., atributo_n)
```

Implementación de formas normales

Para ejemplificar el uso de las formas normales, volvamos a nuestro ejemplo expuesto en la parte de modelado de bases de datos. En este ejemplo poblaremos la tabla con una serie de registros. Nuestro objetivo será la eliminación de redundancias.

Nombre	RUT	Dirección	Hipotecada	Fecha	Sucursal	Dirección
Marcelo	15934234-k	Lota 2332	Si	12/12/2012	Rotonda Atenas	Cuarto Centenario 11232
Fernanda	9238193-2	Choapa 531	No	3/2/2014	Plaza Egaña	Irarrazabal 234
José	10118392-5	Antupiren 8212	No	6/5/2013	Consistorial	Los Presidentes 564
Fernanda	9238193-2	Choapa 531	No	5/2/2014	Rotonda Atenas	Cuarto Centenario 11232
José	10118392-5	Antupiren 8212	No	7/5/2013	Los Leones	Los Leones 4523
Marcelo	15934234-k	Lota 2332	Si	14/2/2014	Rotonda Atenas	Cuarto Centenario 11232
Marcelo	15934234-k	Lota 2332	Si	7/3/2014	Rotonda Atenas	Cuarto Centenario 11232

Imagen 18. Tabla con redundancias.

Los registros incorporados representan a las visitas realizadas por tres clientes en este banco. Cabe destacar que las visitas van a depender de la fecha y la sucursal del banco donde visiten, por lo que cada registro en esta tabla redundante representará una visita específica por un cliente.

Primera Forma Normal (1FN)

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

- Cada campo debe tener un solo valor.
- No pueden haber grupos repetitivos.

Lo que se debe hacer para llegar a esta forma es eliminar aquellos atributos que se encuentren repetidos, y dejar cada repetición como una entrada de una nueva tabla. 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**.

Retomando el ejemplo del banco, nuestra tabla tendría el siguiente punto de inicio:

```
Cliente(nombre, #run, direccion, hipotecada, {fecha, sucursal,
direccion})
```

Nombre	RUT	Dirección	Hipotecada	Fecha	Sucursal	Dirección
Marcelo	15934234-k	Lota 2332	Si	12/12/2012	Rotonda Atenas	Cuarto Centenario 11232
Fernanda	9238193-2	Choapa 531	No	3/2/2014	Plaza Egaña	Irarrazabal 234
José	10118392-5	Antupiren 8212	No	6/5/2013	Consistorial	Los Presidentes 564
Fernanda	9238193-2	Choapa 531	No	5/2/2014	Rotonda Atenas	Cuarto Centenario 11232
José	10118392-5	Antupiren 8212	No	7/5/2013	Los Leones	Los Leones 4523
Marcelo	15934234-k	Lota 2332	Si	14/2/2014	Rotonda Atenas	Cuarto Centenario 11232
Marcelo	15934234-k	Lota 2332	Si	7/3/2014	Rotonda Atenas	Cuarto Centenario 11232

Imagen 19. Punto de inicio.

Cuando vimos el modelo conceptual ya vimos que cada visita debe formar su propio registro en la base de datos, porque no sabemos cuántas visitas va a hacer cada usuario. Los elementos que se consideran como grupos repetidos están encapsulados en el cuadro rojo. En esta tabla, necesitaremos identificar al visitante, por lo que también deberá tener ese atributo. De esta manera, la tabla original se puede desagregar en dos entidades: Cliente y Visita respectivamente.

De esta forma estaremos satisfaciendo la **Primera Forma Normal**:

```
Cliente(nombre, #run, direccion, hipotecada)
```

Nombre	RUT	Dirección	Hipotecada
Marcelo	15934234-k	Lota 2332	Si
Fernanda	9238193-2	Choapa 531	No
José	10118392-5	Antupiren 8212	No

Imagen 20. Ejemplo primera forma normal.

En la entidad Cliente incluimos todos aquellos atributos que dependen exclusivamente del cliente y de nada más.

```
Visita(#run, #fecha, #sucursal, direccion)
```

RUT	Fecha	Sucursal	Dirección
15934234-k	12/12/2012	Rotonda Atenas	Cuarto Centenario 11232
9238193-2	3/2/2014	Plaza Egaña	Irarrazabal 234
10118392-5	6/5/2013	Consistorial	Los Presidentes 564
9238193-2	5/2/2014	Rotonda Atenas	Cuarto Centenario 11232
10118392-5	7/5/2013	Los Leones	Los Leones 4523
15934234-k	14/2/2014	Rotonda Atenas	Cuarto Centenario 11232
15934234-k	7/3/2014	Rotonda Atenas	Cuarto Centenario 11232

Imagen 21. Tabla vista de cliente.

En este caso hemos decidido dejar una clave primaria compuesta por tres elementos (RUT, Fecha y Sucursal), porque cada persona puede hacer varias visitas al banco a lo largo de varios días, y puede ir a varias sucursales en el mismo día.

Esta nueva tabla permitirá asociar cada visita con algún cliente específico mediante el RUT.

Segunda Forma Normal (2FN)

Esta forma debe cumplir las siguientes condiciones:

- Debe satisfacer la 1FN.
- Los atributos deben depender de toda 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 a esta forma debemos fijarnos en las tablas que tengan claves primarias compuestas, porque solo 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. Volviendo a nuestro ejemplo, podemos ver que la dirección de una sucursal depende solamente de la sucursal.

Aplicando la **Segunda Forma Normal**:

```
Cliente(nombre, #run, direccion, hipotecada)
```

Nombre	RUT	Dirección	Hipotecada
Marcelo	15934234-k	Lota 2332	Si
Fernanda	9238193-2	Choapa 531	No
José	10118392-5	Antupiren 8212	No

Imagen 22. Tabla sin modificaciones.

La entidad Cliente no sufre de modificaciones en esta forma normal.

```
Visita(#run, #fecha, #sucursal)
```

RUT	Fecha	Sucursal
15934234-k	12/12/2012	Rotonda Atenas
9238193-2	3/2/2014	Plaza Egaña
10118392-5	6/5/2013	Consistorial
9238193-2	5/2/2014	Rotonda Atenas
10118392-5	7/5/2013	Los Leones
15934234-k	14/2/2014	Rotonda Atenas
15934234-k	7/3/2014	Rotonda Atenas

Imagen 23. Tabla Visita.

La tabla Visita es modificada para separar el atributo dirección dado que depende exclusivamente de cada sucursal.

```
Sucursal(#sucursal, direccion)
```

Sucursal	Dirección
Rotonda Atenas	Cuarto Centenario 11232
Plaza Egaña	Irarrazabal 234
Consistorial	Los Presidentes 564

Imagen 24. Tabla Sucursal.

La nueva tabla Sucursal generada permitirá asociar la dirección de cada sucursal, utilizando el nombre de ésta para generar una clave de referencia. Con esto nos ahorramos todo el espacio que implicaría guardar la dirección de la sucursal cada vez que nos visite un cliente.

Tercera Forma Normal (3FN)

Esta forma debe cumplir las siguientes condiciones:

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

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**.

Hasta el momento, nuestra tabla de Cliente presenta la columna Hipotecada. Resulta que para cumplir con la tercera forma normal, debemos aclarar este comportamiento. La tercera forma normal nos indica que el status de Hipoteca de una propiedad no depende del cliente, si no de la propiedad. De esta forma, al separar la hipoteca y asociarla con la propiedad estamos evitando instancias de dependencia funcional transitiva entre la hipoteca y el cliente. En este caso, si una propiedad está hipotecada depende de qué propiedad hablamos, pero no del cliente que va al banco.

Aplicando la **Tercera Forma Normal**:

```
Cliente(nombre, #run, direccion)
```

Nombre	RUT	Dirección
Marcelo	15934234-k	Lota 2332
Fernanda	9238193-2	Choapa 531
José	10118392-5	Antupiren 8212

Imagen 25. Ejemplo Tercera forma normal.

Separamos el atributo hipoteca del cliente y posteriormente lo agregamos a una nueva tabla llamada Propiedad. Resulta que para asociar la hipoteca con el cliente a nivel de tablas, necesitamos vincular la dirección como clave primaria.

```
Propiedad(#Direccion, hipotecada)
```

Dirección	Hipotecada
Lota 2332	Si
Choapa 531	No
Antupiren 8212	No

Imagen 26. Tabla Propiedad.

```
Visita(#run, #fecha, #sucursal)
```

RUT	Fecha	Sucursal
15934234-k	12/12/2012	Rotonda Atenas
9238193-2	3/2/2014	Plaza Egaña
10118392-5	6/5/2013	Consistorial
9238193-2	5/2/2014	Rotonda Atenas
10118392-5	7/5/2013	Los Leones
15934234-k	14/2/2014	Rotonda Atenas
15934234-k	7/3/2014	Rotonda Atenas

Imagen 27. Tabla Visita tercera forma normal.

```
Sucursal(#sucursal, direccion)
```

Sucursal	Dirección
Rotonda Atenas	Cuarto Centenario 11232
Plaza Egaña	Irarrazabal 234
Consistorial	Los Presidentes 564

Imagen 28. Tabla Sucursal.

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. En nuestro ejemplo, si quisiéramos cambiar la dirección de una sucursal, no tendríamos que cambiar el dato en cada una de las visitas, sino que basta con modificar el registro correspondiente a dicha sucursal.

El lado negativo de la normalización, es que pueden incrementar los tiempos de respuesta. En este caso, si queremos saber cuántos de los clientes que fueron a una sucursal cierto día tienen su vivienda hipotecada, tendríamos que consultar la tabla Visitas, con el resultado ir a cliente, y con ese resultado buscar en la tabla propiedades.

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

Tabla Paciente

Paciente	Tratamiento_Id	Doctor_Id
María Zenteno	1	1
Fernando Sánchez	2	2
José Artigas	1	2
Emilia Tapia	1	1
Felipe Zamorano	2	1
César Oyarce	2	1
Catalina Maillard	1	2

Imagen 29. Tabla Paciente.

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.

Tabla Doctores

id	Nombre
1	Astorquiza
2	Feris

Imagen 30. Tabla doctor.

Tabla Tratamientos

id	Nombre
1	Control Médico
2	Biopsia

Imagen 31. Tabla tratamientos.

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 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.

Tabla desnormalizada que permite asociar cada paciente con un doctor y un tratamiento específico

Paciente	Tratamiento_Id	Tratamiento_Nombre	Doctor_Id	Doctor_Nombre
María Zenteno	1	Control Médico	1	Astorquiza
Fernando Sánchez	2	Biopsia	2	Feris
José 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 32. Tabla desnormalizada.

En el contexto de la ciencia de datos, generalmente trabajamos con una sola tabla de datos. En los casos que tengamos una base de datos con múltiples tablas, nuestro trabajo será desnormalizarla mediante la concatenación de registros y columnas en una nueva tabla procedimental.

¿Y por qué debemos desnormalizar?

De manera simple, todo modelo o procedimiento estadístico que deseemos implementar debe surgir de una misma matriz de datos. El paso de desnormalizar nos obliga a tener un problema definido en un conjunto de registros de entrenamiento/prueba y una serie finita de atributos. Casos como los estudios de serie de tiempo, los estudios longitudinales panel y los estudios mediante ecuaciones multinivel hacen uso explícito de estructuras de datos desnormalizadas.