Diseño de datos

Tabla de contenido

Datos

Diseño de archivos y bases de datos

Los archivos y las bases de datos son los que nos permite hacer que la información que utilizan las aplicaciones que manipulan los sistemas puedan persistir. Nos ayudan a que las aplicaciones recuerden los datos que van a utilizar en una próxima sesión.

La información persistente que los sistemas tienen que recordar se implementa con:

- ☐ Memoria
- ☑ **Archivos**
- ☑ **Bases de datos**
- ☐ PaaS (Platform as a Service)

Tres niveles de modelado

En general, cuando hablamos de modelado de datos nos vamos a encontrar en la literatura con 3 niveles:

1) Un nivel que tiene que ver más con lo **conceptual**, que es ni más ni menos lo que hemos visto con el modelo de dominio. El modelo de dominio es un modelo conceptual de datos, nos muestra cuáles son las entidades y las relaciones entre esas entidades que son importantes para el dominio del problema que estamos estudiando. Ese modelado conceptual, como su nombre lo indica, no tiene absolutamente nada que ver con el diseño ni con la implementación. Simplemente estamos modelando conceptos que forman parte del dominio que estamos analizando.
2) Después podemos, a través de algunas transformaciones de ese modelo conceptual, obtener un diseño **lógico**. El diseño lógico o el modelo lógico tiene que ver con transformar esas entidades, esas relaciones del modelo de dominio, en un diseño lógico de los archivos y de la base de datos que vamos a utilizar. En definitiva, vamos a identificar: cuáles son las tablas; cuáles son los archivos planos; cuáles son los atributos, los campos; cuáles son las columnas, en el caso de una base de datos relacional; cuáles de esos atributos o columnas son claves primarias, cuáles claves foráneas; tenemos que implementar las relaciones.
3) El último nivel de modelado que nos vamos a encontrar es el **físico** que nos dice que esas bases de datos que has identificado, esas tablas, hay que trasladar eso a la implementación propiamente dicha. Así como traducimos el diseño de una clase a la implementación de la clase, acá tenemos que hacer lo mismo: Tenemos un modelo conceptual que hemos transformado en un modelo lógico; y ahora tenemos que traducirlo un modelo físico. ¿Y eso qué implica? Bueno, generar las sentencias necesarias para crear las tablas, para crear los índices, para definir los espacios de almacenamiento que va a utilizar la base de datos, para definir aspectos de seguridad, etc.

[V / **F**] El nivel de modelado conceptual está profundamente relacionado con el diseño y la implementación.

Diseño de archivos y bases de datos

Hablábamos hace un rato que, en definitiva, la persistencia de los datos se hace a través de archivos (que ustedes ya conocen) o a través de bases de datos. ¿Qué es una base de datos? Es una colección ordenada de datos administrada por un sistema de gestión, usualmente conocido como DBMS (por las siglas de Data Base Management System). Eso fue una especie de aplicación que está en background corriendo. Ustedes van a poder: "quiero recuperar los datos del alumno tal"; este sistema se va a encargar de traducir esa consulta en términos que puedan ser entendidos por el motor de base de datos (el sistema de gestión de base de datos normalmente se conoce como motor de base de datos); ese motor de base de datos va a realizar las consultas; nos va a traer los datos que necesitamos; y nos lo va a dar. Eso en definitiva es lo que a grandes rasgos nos resuelve un motor de base de datos. Por supuesto que esto es una introducción, no vamos a profundizar en este tema. Hay una materia entera que habla de base de datos. Nuestra idea es darles algunos lineamientos como para que puedan ubicarse con el trabajo práctico que tienen que realizar.

Modelos de bases de datos

Entonces, lo que tenemos es un sistema de base de datos que nos ayuda a administrar la base de datos. Normalmente lo que nos vamos a encontrar es que para una aplicación vamos a tener una base de datos, o varias; normalmente en una organización, varias aplicaciones van a compartir 1, 2 o 3 bases de datos. Seguramente conocerán algunos de los proveedores/marcas de base de datos más conocidas como Oracle, IBM, IBM DB2. También algunas Open Source o de disponibilidad pública como MySQL, PostgreSQL, etc. Históricamente los modelos de base de datos que han habido en los últimos 30/40 años, han sido: el **jerárquico**; el modelo en **red**; el **relacional**; cuando apareció objetos, aparecieron las bases de datos orientadas a **objetos**; y hoy también estamos hablando de base de datos no estructuradas, orientadas a **documentos** como puede ser MongoDB y algunas otras que andan dando vueltas por ahí.

Tipos de Modelos de Bases de Datos:

- ☑ **De red**
- ☐ De dominio
- ☑ **De objetos**
- ☑ **Orientado a documentos**
- ☑ **Relacional**
- ☑ **Jerárquico**

A fines de los 60, principios de los 70 aparece el primero, el **modelo jerárquico**, que lo que nos planteaba era… Bueno, vos en general, te vas a encontrar en tu modelo conceptual con estructuras del tipo cabecera-detalle. El modelo de base de datos jerárquico lo que busca justamente es implementar ese tipo de cosas. Por ejemplo, en una factura, ustedes han modelado hasta ahora un pedido o una operación de compra, que en general tiene una cabecera con los datos generales y después tiene un detalle con todos los ítems que están incluidos en una factura o en un pedido. Bueno, la base de datos jerárquica sigue ese esquema: un master y un detail.

En las **bases de datos en red** el esquema es también similar; la diferencia es que por ahí uno en una base de datos jerárquica tiene que navegar la jerarquía, mientras que en una base de datos en red uno puede navegar de un registro al otro.

En una **base de datos relacional**, lo que tenemos son tablas. Son tablas con filas o distancias que están relacionadas entre sí con otras tablas.

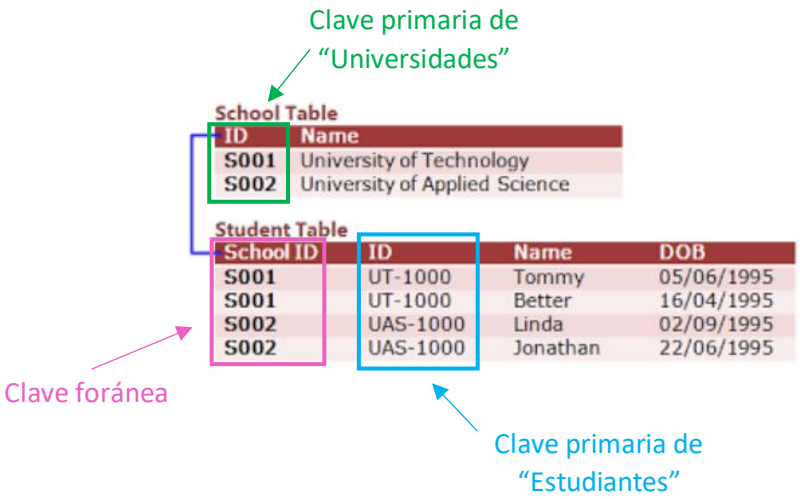
Y, más recientemente, tenemos la base de datos objeto, como decíamos, y las bases orientadas a documentos.

[**V** / F] Las tablas son elementos claves en los modelos de bases de datos relacionales.

Bases de datos relacionales

La idea de la base de datos relacional viene de Edgar Codd de a finales de los 60. Esto explotó a finales de los 80 muy fuertemente y los 90 fue la época de las bases de datos relacionales. Ahí es cuando surge Oracle, Microsoft le sigue atrás con SQL Server, más tarde aparecen las base de datos de código abierto… Ha sido, en definitiva, el caballito de batalla y es el modelo que ha ganado de alguna manera, que se ha popularizado. El modelo de base de datos relacional lo que plantea es que vamos a tener en efectiva tablas; cada tabla tiene columnas; esas columnas, en definitiva, son los atributos; y también hay para distinguir cada una de las instancias…

Ahí lo que estamos viendo es un ejemplo de una tabla de "Universidades" (con la universidad tecnológica y la universidad de ciencias aplicadas) y abajo lo que tenemos es una tabla de "Estudiantes" (donde tenemos distintos alumnos). Acá aparecen varios conceptos importantes:



Por un lado aparece la idea de la **clave primaria**. La clave primaria es lo que permite identificar a las distintas filas de forma unívoca. Entonces este atributo/clave ID es la clave primaria de la tabla "Universidades". A su vez, la clave primaria de alumnos es esta otra columna que vemos acá que tiene un pésimo nombre que es ID. Eso en realidad, son la identificación de cada estudiante.

Y lo que aparece a la izquierda de ID en la tabla de "Estudiantes" es la **clave foránea**. La clave foránea es lo que nos permite implementar la relación entre una tabla y la otra. Fíjense como se parece esto a los objetos y las relaciones o asociaciones del modelo conceptual, del modelo de dominio. Acá lo que estamos haciendo en definitiva es implementar las asociaciones como atributos que referencian a instancias de otras tablas, cosa que no pasa en la jerárquica ni en la base de datos en red. Ahí son punteros, de la misma manera que hay punteros entre objetos. Acá lo que aparece es explícitamente un atributo, una columna que tiene la clave primaria de otro objeto, en definitiva esa es la clave foránea, y lo que permite implementar la relación.

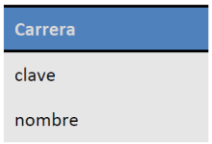
[V / **F**] Las tablas primarias son necesarias para identificar las claves foráneas.

Bases de datos relacionales

Tablas, columnas y claves

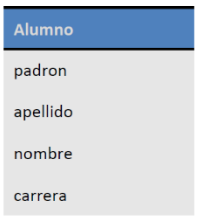
Entonces, en general, lo que nos vamos a encontrar es que…

Por ejemplo, para una entidad del modelo de dominio "Carrera" lo que vamos a tener probablemente es una tabla llamada *carrera* con 2 columnas que en definitiva van a representar los atributos del objeto de dominio "Carrera". Acá ya le estamos indicando que *clave* es un INT y que *nombre* es un CHAR de 20; y adicionalmente estamos diciendo que *clave* es una clave primaria. Adicionalmente, yo después voy a poder crear un índice sobre esa clave primaria, que es lo que me va a permitir ubicar cada una de las instancias, cada una de las filas de esta tabla.




```
CREATE TABLE carrera (
        clave INT PRIMARY KEY,
        nombre VARCHAR(20)
)
```

Siguiendo con el ejemplo, yo debería poder tener una tabla *alumno* que implemente el objeto de dominio "Alumno" con: *padrón*; *apellido*; *nombre*; *carrera*; y acá lo que estamos diciendo es que tenemos un *FOREIGN KEY(carrera)* que referencia a *carrera(clave)*. Esto es lo que nos permite implementar la relación entre "Alumno" y "Carrera".




```
CREATE TABLE alumno (
        padron INT PRIMARY KEY,
        apellido VARCHAR(20),
        nombre VARCHAR(20),
        carrera INT,
        FOREIGN KEY(carrera) REFERENCES carrera(clave)
)
```

"Yo soy alumno de una carrera". Acá voy a tener un puntero, en definitiva, un atributo *carrera* (el de *carrera INT*) que va a estar apuntando (por el atributo clave de …*REFERENCES carrera(clave)*) a la columna *clave* de la tabla *carrera*. Con eso yo implemento la asociación entre una y otra.

Restricción que tiene esto: el alumno puede estar anotado solamente en una carrera, no puede estar cursando más de una carrera. Ahí hace falta otro truco para resolver ese tema.

Del modelo de Dominio a la Base de Datos

Transformaciones

Si yo quiero transitar el camino del modelo de dominio a la base de datos, ¿qué transformaciones puedo hacer? Bueno, hay algunas bastantes clásicas:

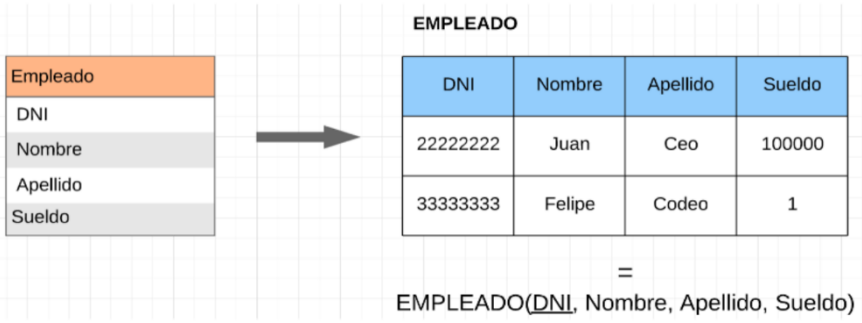
- Una es implementar cada objeto de dominio como una tabla. Sería el ejemplo que acabamos de ver: una tabla por cada objeto de dominio.
- En el caso de que tengamos en el modelo de dominio superclases y subclases, bueno, ahí hay que hacer alguna cosita en particular que la vamos a explicar también un poquito más adelante. Se va a poder implementar como una tabla para superclases y subclases o una tabla para cada una.
- Y después está el tema de que hay que implementar las asociaciones: Por eso nosotros insistimos tanto en el modelado de dominio que una cosa son los atributos y otra cosa distinta son las asociaciones entre entidades. Con esto lo que queremos decir es que en el modelo de dominio, uno no tiene atributos que referencien a la clave de otros objetos porque estamos hablando de un modelo conceptual. Si yo ese modelo lo fuera a trasladar a un modelo jerárquico, lo voy a implementar de una manera; si fuera a transformar ese modelo de dominio para poder tener una base de datos relacional, voy a tener que hacer otro tipo de transformaciones y voy a tener que implementar este tema de las claves primarias y foráneas, y voy a tener que implementar las asociaciones como claves foráneas.

Opciones de transformaciones de entidades del modelo de dominio al modelo de una base de datos relacional son:

- ☑ **Una tabla para cada superclase y otra tabla para cada subclase**
- ☑ **Una sola tabla para cada superclase y subclases**
- ☑ **Una tabla por cada objeto de dominio**

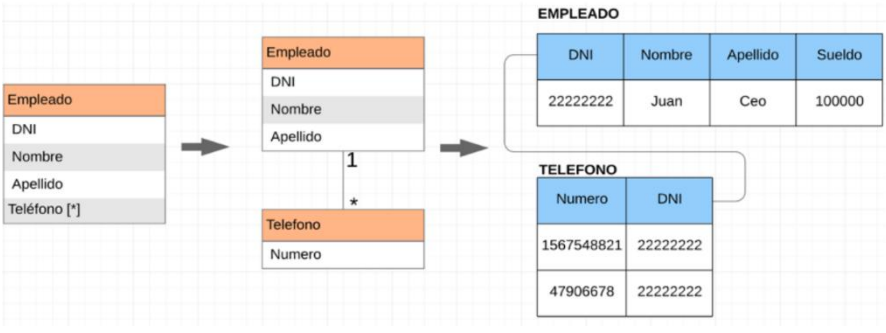
Vamos a ver un poquito una por una estas transformaciones:

Cuando tengo esta idea entonces de transformar por cada objeto de dominio tener una tabla en el modelo relacional. En este caso, lo que tenemos en el modelo de dominio un objeto de negocio "Empleado" con DNI, Nombre, Apellido y Sueldo. Probablemente le podría haber puesto entre llaves al costado de DNI la indicación de que es un identificador único natural para empleados ("{id}"). Cuando yo transformo esto en una tabla, lo que voy a tener es una tabla llamada *EMPLEADO* con las columnas *DNI*, *Nombre*, *Apellido* y *Sueldo*. Cada una de esas columnas va a tener su propio tipo de datos y *DNI* va a ser la clave primaria.



Atributos multivaluados

Ahora fíjense que yo en el modelo de dominio de repente puedo tener un atributo repetitivo como Teléfono. El empleado puede tener varios teléfonos: el teléfono de su casa, el teléfono celular… y eso yo no lo voy a representar normalmente en el modelo de dominio de esta forma (el modelo del medio); sino que lo que voy a tener que hacer es tener acá un atributo para el cual estoy, indicando que puedo tener varias ocurrencias ("Teléfono [*]"). Cuando yo transformo esto a una base de datos tengo que "normalizar", tengo



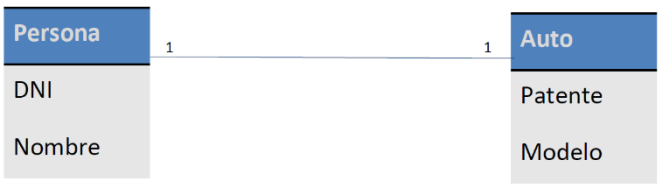
que sacar ese grupo repetitivo. No puedo tener en una tabla un array, no puedo tener un vector con cada uno de los teléfonos, lo tengo que tener una tabla aparte. Entonces aparece una tabla, aparte de la tabla *EMPLEADO*, la tabla *TELEFONO* con cada uno de los números de teléfono de un empleado en particular. Y necesito sí o sí *DNI* como clave foránea para poder apuntar a qué empleado corresponde el número de teléfono, sino no tengo manera de resolver la asociación.

Esta es una licencia, si se quiere "poética", porque lo que estamos tratando de representar acá es que yo podría agarrar y decir… bueno, a ver, teléfono es como si fuera un objeto de dominio aparte, cosa que no recomendamos. El teléfono, en general, es un atributo. Nos tomamos la licencia poética acá para que entiendan que para un empleado va a haber varios teléfonos. Entonces, eso lo tenemos que implementar como 2 tablas diferentes en la base de datos.

Así que recuerden, cuando ustedes encuentren que hay un atributo en un objeto que puede tener múltiples ocurrencias, lo que hay que hacer es **normalizar**. Normalizar significa, entre otras cosas, sacar los atributos que son repetitivos y ponerlos en una tabla aparte.

Relaciones 1 a 1

¿Qué hacemos cuando tenemos asociaciones o relaciones 1 a 1? Una persona tiene un auto. Bueno, acá lo que hacemos es tenemos una tabla para PERSONA, una tabla para AUTO, *DNI* y *patente* son las claves primarias, y lo que tenemos que resolver es de qué lado ponemos la columna que nos permite implementar esta relación que estamos poniendo acá. Entonces, normalmente, se toma del lado del objeto que más peso o sentido tenga. ¿Qué es más importante en este esquema: el auto o la persona? Bueno acá quien diseñó la base de datos optó por decir… bueno, a ver, la tabla PERSONA va a incluir un atributo, una clave, que va a apuntar a AUTO. Es una clave foránea. En definitiva, **PatenteAuto** va a estar apuntando a la clave primaria de AUTO. Pero podría haber sido al revés, tranquilamente. Fíjense que en este ejemplo en particular, la multiplicidad es 1 a 1. Es decir que no hay condicionalidad en ninguno de los 2 lados. Con lo cual, la clave foránea tiene que sí o sí tener un valor. No puede tomar valores nulos.

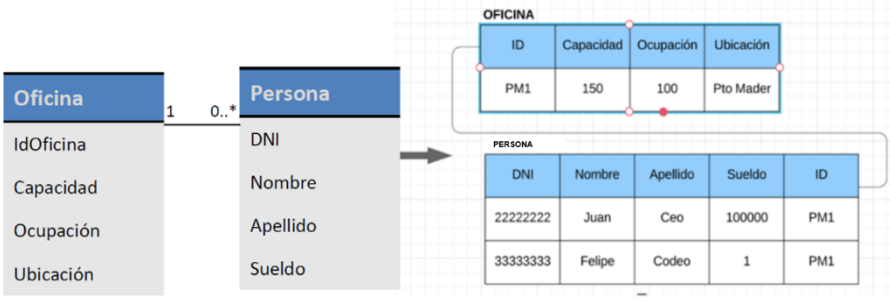


PERSONA(DNI, Nombre, **PatenteAuto**)
AUTO(Patente, Modelo)

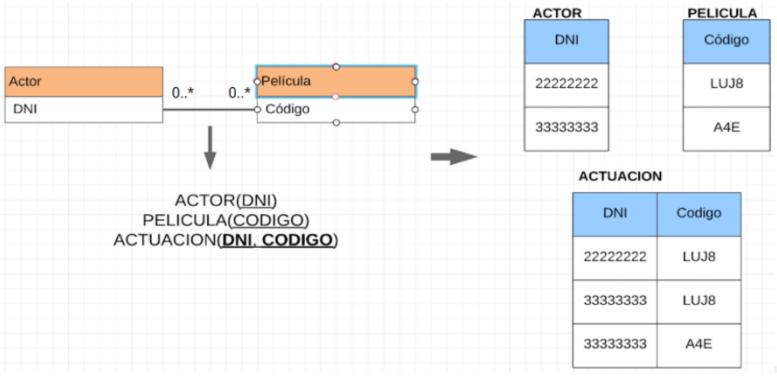
PERSONA(DNI, Nombre)
AUTO(Patente, Modelo, **DNI**)

Relaciones 1 a 0..*

Aquí tenemos otro ejemplo: Una persona que trabaja en una oficina, y en una oficina pueden trabajar cero o varias personas. ¿Eso cómo se representa? Como dijimos antes, una tabla por cada objeto, y normalmente lo que hacemos es en la tabla que implementa al objeto de conectividad 0..* le incluimos el ID de la tabla que implementa el objeto de conectividad 1.



Relación 0..* a 0..*



Cuando tenemos una asociación muchos a muchos, yo había comentado hace unos minutos, que había que hacer alguna cosita un poquito distinta. Aparece algo que se llama **tabla de correlación**. Yo tengo que conectar una instancia con muchas instancias del otro lado y, a su vez, una instancia de acá con muchas instancias del otro lado. Entonces, claramente en una película puede haber muchos actores y un actor puede estar en muchas películas. Eso a nivel modelo conceptual está perfecto. Ahora, cuando yo necesito trasladar eso a una base de datos relacional, necesito que esta asociación (la de 0..* a 0..*) se transforme en una tabla distinta. Entonces acá tenemos la tabla *ACTOR* con su *DNI*, la tabla *PELICULA* con su *Código* y en la tabla *ACTUACION* tenemos el/la listado/combinación de qué actores han estado en cada una de las películas. Eso no lo puedo resolver poniendo claves foráneas ni en *ACTOR* ni en *PELICULA* porque tendría que poner una película apuntando a muchos actores (que es variable) y entonces tendría que tener una columna por cada actor, lo cual no tiene sentido; y en actor tendría que tener una columna con cada película, y, a medida que se vayan agregando películas voy a tener que agregar una columna. ¡No tiene ningún tipo de sentido! Por eso aparece la tabla de correlación.

¡Ojo! También acá hay que tener cuidado con el tema del cero en la relación **0**..*. Acá, simplemente, si son 0..* y 0..*, lo que va a terminar pasado es que algunas instancias en *ACTUACION* no van a existir.

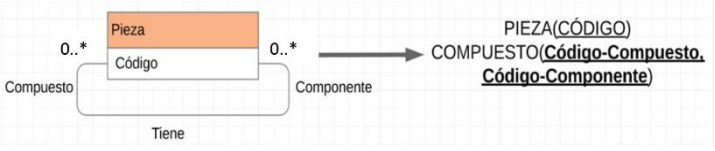
Recursivas 0..1 a 0..1

Las que son recursivas, estas que son reflexivas. Por ejemplo, Categoría (de producto) del ejercicio 4.6. Siempre hemos visto que tienen que ser condicionales, porque si no en el modelo de dominio esto se convierte en algo recursivo. Entonces imaginemos que una persona está casada con otra, lo que tenemos que tener es el DNI (por ahí faltaría el nombre). Yo debería poder implementar esta asociación que estamos viendo acá. Entonces lo que agregamos es una columna, que es en definitiva clase foránea, que va a estar apuntando al DNI de otra persona con la cual estoy casada (DNI-CÓNYUGE). Nuevamente, este valor va a poder tomar valores nulos. Esta columna va a poder tomar valores nulos, porque no todo el mundo está casado; y por supuesto, ni que hablar si esta persona se separa y se casa con otra. Este modelo no lo permite. Por supuesto hay que cambiar el DNI acá del cónyuge. Pero, si nos interesara resguardar la historia, con esta estructura no lo puedo hacer. Es otro requerimiento.

Recursivas 0..1 a 0..*

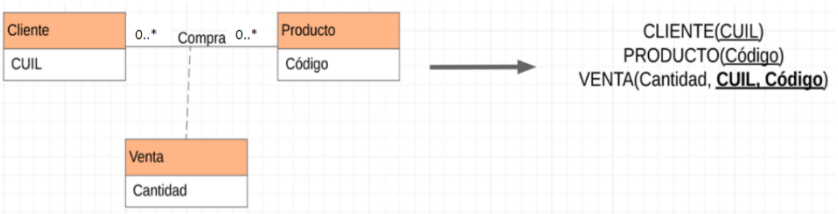
Estamos ante la misma situación. Vamos a tener que agregar una columna que implemente la relación recursiva. Acá la diferencia quizás es que en el caso anterior teníamos 0..1 a *0..1* y ahora tenemos de 0..1 a **0..***. En definitiva, es la categoría apuntando a su categoría superior (si pensamos en el ejercicio 4.6). Acá es la persona apuntando a su madre o alguno de sus progenitores.

Recursivas 0..* a 0..*



Misma situación. Necesitamos una tabla de correlación como el caso que vimos antes.

Clases asociativas

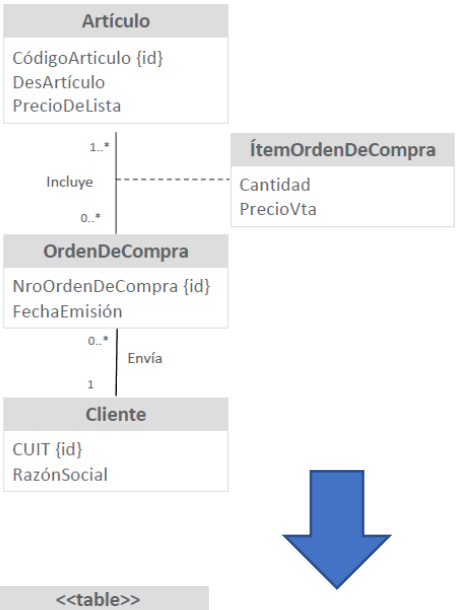


Las asociativa fácilmente se transforman. En definitiva, lo que termina pasando es que la clase asociativa (en este caso VENTA) termina teniendo como claves los atributos de los elementos que asocia (*CUIL*, *Código*). Entonces, por eso, cuando nosotros les decíamos "los objetos asociativos son asociaciones Y objetos al mismo tiempo", lo que decíamos era no pongan acá los atributos identificadores de estos 2 objetos (los que quedan asociados), porque no hace falta; ya sabemos que va a haber una instancia de Venta por cada combinación de Cliente con Producto. Entonces aquí lo que aparece es, en la implementación, una tabla para CLIENTE, una tabla para PRODUCTO y una tabla para implementar la asociación. Es la misma situación, o una situación similar, a la que se planteaba antes con la tabla de correlación. La diferencia que la tabla de correlación solamente tiene como columnas las claves de las 2 tablas que está conectando. Acá lo que tenemos es además un atributo propio, que en este caso es Cantidad.

Dicho sea de paso, en este ejemplo puntualmente, acá yo puedo vender solamente una vez a un cliente el mismo producto. No puedo vender más de una vez el mismo producto al mismo cliente. Es una limitación de este modelo de dominio.

Datos

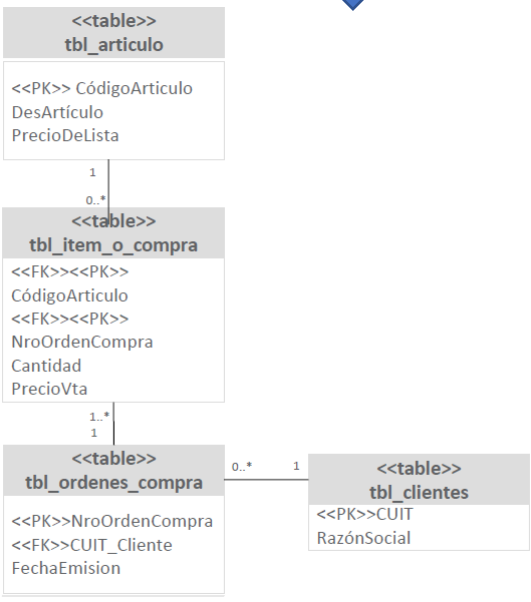
Del modelo de dominio a la base de datos relacional



Acá tenemos un ejemplo de una estructura clásica:

El cliente hace una orden compra; la orden de compra está compuesta por varios ítems de orden de compra que están relacionados con un artículo. Entonces ahí lo que tenemos es un objeto de dominio Artículo que está asociado a un objeto OrdenDeCompra; de la combinación de los 2 surge el objeto asociativo ItemOrdenDeCompra; y tenemos el objeto Cliente; y tendríamos que hacer la transformación de esto al modelo relacional. Por lo que decíamos recién, deberíamos tener una tabla por cada objeto en principio; y tendríamos que resolver el tema de las asociaciones.

Fijémonos cuál es la transformación que se produce:

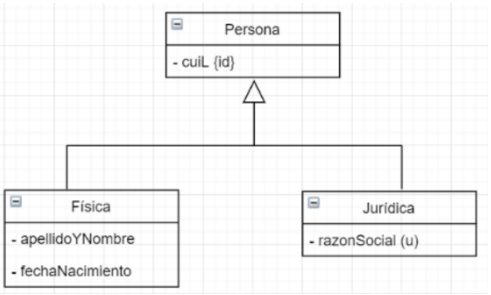


Acá estamos utilizando una variación de UML que está pensada para modelar bases de datos. Fíjense que acá los objetos ahora están representando tablas: Cuando agregamos el estereotipo de arriba <<table>> estamos indicando que son tablas. Artículo y OrdenDeCompra se han transformado en tbl_articulo, tbl_item_o_compra, tbl_ordenes_compra y tbl_clientes. Lo que pasa normalmente en las organizaciones es que hay estándares para ponerle nombre a las tablas, a las columnas. Entonces hay que seguir esos lineamientos. Acá fíjense que entre Artículo y tbl_articulo lo único que tenemos es que hemos recuperado el CódigoArtículo, DesArtículo y PrecioDeLista que son, en este caso, el estándar que se utiliza para nombre de columnas es igual a lo que tenemos acá en los atributos del objeto de dominio. CódigoArtículo que era antes un {id} ahora se ha transformado en una Primary Key (en este caso en particular, en esta variación de la notación UML, cuando queremos indicar que una columna es Primary Key le agregamos el prefijo: <<PK>>). Después lo que tenemos es que ItemOrdenDeCompra se ha transformado en una

tabla que tiene como columnas Cantidad y PrecioVta, pero, a su vez, tiene 2 columnas adicionales que son al mismo tiempo Primary Key las 2 combinadas y Foreign Key cada una por separado. <<FK>><<PK>>CódigoArtículo es Foreign Key y forma parte de la Primary Key. Quiere decir que, con este código CódigoArtículo resuelvo la asociación para el lado del Artículo; y acá tengo el <<FK>><<PK>>NroOrdenCompra que me permite vincular el detalle de la tbl_ordenes_compra con la cabecera. Adicionalmente, esta asociación Cliente con OrdenDeCompra como es una asociación 1 a 0..*, la resolvimos incluyendo en tbl_ordenes_compra el <<FK>>CUIT_cliente (que es el <<PK>>CUIT de tbl_clientes).

Después, lo que hay que acordarse es de normalizar las estructuras de esos atributos que en el modelo de dominio son repetitivos, normalizarlos y ponerlos en una tabla aparte.

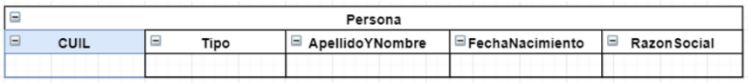
Herencia



Si ustedes llegaron a usar una estructura de generalización-especialización en el modelo de dominio, lo que habrá que resolver acá es qué es lo que pasa. En esta estructura, lo que tenemos es que Persona es un objeto de dominio que tiene el CUIL (que es identificador) y si es Persona Física, va a tener apellidoYNombre y si es Persona Jurídica, razónSocial. Entonces, acá lo que hay que decidir es ¿qué hacemos?: Si mezclamos las 2 subclases y el padre en una sola tabla, o si tenemos una tabla por cada uno de los hijos. Bueno, vamos a ver qué alternativas tenemos.

Supresión de subentidades

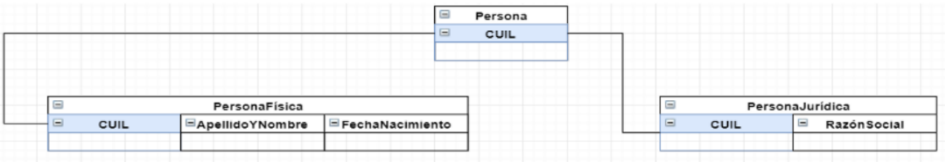
Persona(CUIL, tipo(F|J), (apellidoYNombre), (fechaNacimiento), (razónSocial))



La primera consiste en agarrar y decir, bueno combino todo en una sola tabla para no andar complicándome. Tengo una sola tabla Persona que representa tanto personas físicas como personas jurídicas y lo que tengo que tomar en cuenta es que, por ejemplo, si estoy dando de alta una persona física, ahí tengo una columna que me permite decidir si este registro corresponde a una física o a una jurídica (tipo(F|J)). Si es física, tendrá que tener apellidoYNombre y fechaNacimiento; y si es jurídica tendrá que tener razónSocial. Entonces, en este caso algunos datos serán nulos y otros no, hay que tener cuidado con eso. Esta es la solución más sencilla.

Derivación a tablas de todas las entidades

Persona(CUIL)
PersonaFísica(**CUIL**, apellidoYNombre, fechaNacimiento)
PersonaJurídica(**CUIL**, razónSocial)

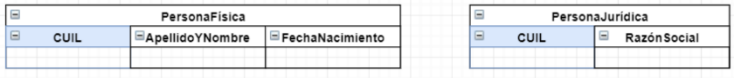


La otra alternativa es implementar una tabla por la cabecera, y una tabla por cada una de las de las variantes de las subclases. Siempre estamos hablando de subclases y superclases conceptuales en el modelo de dominio. Entonces acá lo que tenemos es: para la cabecera (en este caso Persona) implementamos una tabla para esto (la tabla Persona), una tabla PersonaFísica, una tabla PersonaJurídica (es decir una tabla para cada hijo) y lo que tengo que resolver acá es **CUIL** para saber cómo navegar de un lado para el otro. Acá pareciera que no tendría mucho sentido porque persona en sí mismo, no tiene un atributo propio, no tiene columnas propias. Entonces uno se pregunta ¿para qué lo necesito? A veces por temas de diseño, si yo necesito recorrer todas las personas físicas y jurídicas por CUIL y tengo que buscarlas… bueno, es mucho más fácil, por ahí, recorrer Persona y después ir a PersonaFísica o ir a PersonaJurídica. Es un tema de diseño y esto algún DBA (Data Base Administrator) los va a guiar.

Supresión de entidad general

Y la otra alternativa es olvidarse del padre y dejar solamente la PersonaFísica y la PersonaJurídica. Esto claramente no sería aplicable en el ejemplo que tenemos si la cabecera Persona tuviese algún otro atributo además del CUIL. Pero bueno, si lo hubiera, este esquema no serviría porque nos perdemos ese dato; y, en todo caso, si es común, habría que repetirlo en cada una de estas 2 tablas, con lo cual, estaríamos incurriendo en una redundancia.

Física(CUIL, apellidoYNombre, fechaNacimiento)
Jurídica(CUIL, razónSocial)



Normalización

El otro tema que mencionábamos recién era el de la normalización. Lo que vimos recién es que al sacar el teléfono de adentro de empleado o de adentro de alumno, lo que estamos haciendo es llevar eso a primera forma normal. Esto originalmente viene de cuando se empieza con el tema de archivos planos. En general lo que había eran grandes conjuntos de datos sin normalizar. Entonces, en definitiva, uno tenía registros que tenían información que estaba repetida. Entonces, de repente, uno tenía una fila o un registro para cada uno de los ítems de compra, por ejemplo. Y se repetía el número de factura, la fecha de la factura, el cliente, la razón social, etc. para cada ítem. Entonces lo que se planteaba era… bueno no podemos tener esa redundancia de datos, apliquemos normalización. La normalización surge como la gran herramienta para poder estructurar esos datos. Lo que pasa es que en paralelo con eso, históricamente, surge el modelado conceptual; y el modelado conceptual ya te plantea un esquema mucho más ordenado, mucho más, si se quiere, si bien no apunta a que sea algo así implementable tal cual, pero ya te planea entidades, te plantea objetos entonces como que ya te deja un esquema armado. Entonces el creador, digamos, de algo que fue previo al modelo de dominio que ustedes hacen, que es el diagrama de entidad-relación, justamente planteaba, bueno, había como una lucha ahí entre normalización y el diagrama de entidad-relación. ¿Por qué? Porque el resultado que uno obtiene de un modelo conceptual hace que, en general, uno no tenga que aplicar la técnica de normalización para eliminar redundancia de datos, salvo algunos casos puntuales como estos que mencionábamos recién (por ejemplo, tener algún atributo que sea repetitivo).

Así que, en general, nos vamos a encontrar, en la normalización, con que:

- En la **primera forma normal** (1FN) lo que se busca es eliminar los atributos repetitivos, entonces lo que hacemos es llevar eso a otra tabla.
- En la **segunda forma normal** (2FN) lo que se busca es que en los registros que tienen claves compuestas, todos los atributos que forman parte de ese registro sean funcionalmente dependientes de toda la clave. Con lo cual, si hay algún atributo que no es dependiente de los elementos que forman parte clave hay que eliminarlo.
- En la **tercera forma normal** (3FN) todos los atributos son funcionalmente dependientes de la clave.

Ejemplo

Tabla sin normalizar

ClienteID	Nombre	Localidad	CostoTransporte	ArtículoID	Artículo	Cantidad	Fecha
11	Luis	Suba	50.000	A1	Papel	100	3/5
11	Luis	Suba	50.000	A3	Cinta	50	5/5
11	Luis	Suba	50.000	A9	Lápiz	200	7/5
44	Ana	Centro	65.000	A1	Papel	100	10/5
55	José	Puente Aranda	70.000	A4	Grapas	30 50	3/5 5/5

Acá hay una tabla sin normalizar: Ahí tenemos repetido el cliente, el nombre, la localidad, el costo del transporte, el artículo, la cantidad, la fecha. Entonces, para transformar esto en una primera y segunda forma normal, lo que se hace es: Si nos fijamos acá tenemos: datos que tienen que ver con el cliente.; Nombre es funcionalmente dependiente del cliente; Localidad es funcionalmente dependiente del cliente.

Primera y segunda forma normal

Clientes

ClienteID	Nombre	Localidad	CostoTransporte
11	Luis	Suba	50.000
44	Ana	Centro	65.000
55	José	Puente Aranda	70.000

Artículos

ArtículoID	Artículo
A1	Papel
A3	Cinta
A4	Grapas
A9	Lápiz

Ventas

ClienteID	ArtículoID	Cantidad	Fecha
11	A1	100	3/5
11	A3	50	5/5
11	A9	200	7/5
44	A1	100	10/5
55	A4	30	3/5
55	A4	50	5/5

Bueno, entonces acá se le aplica estas técnicas que mencionábamos recién y nos queda por un lado el Cliente, por otro lado las Ventas y por otro lado los Artículos. Todavía tenemos ahí una tercera forma normal porque el costo del transporte no va a depender tanto del cliente, sino que pareciera que tuviera sentido tener una tabla con las localidades y los costos de transporte para cada una de esas:

Tercera forma normal

Clientes

ClienteID	Nombre	Localidad
11	Luis	Suba
44	Ana	Centro
55	José	Puente Aranda

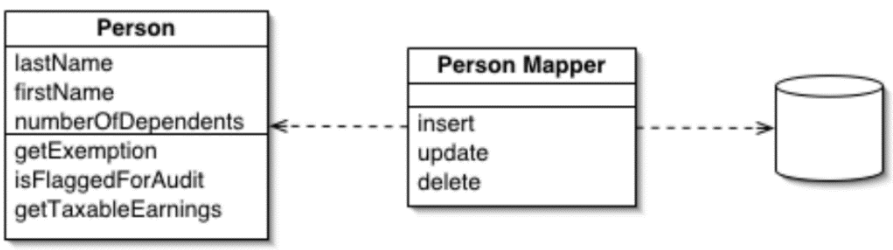
Transporte

Localidad	CostoTransporte
Suba	50.000
Centro	65.000
Puente Aranda	70.000

Normalmente lo que uno se encontraba hace mucho tiempo atrás, eran grandes masas de datos que había que normalizar/estructurar. Aplicando la técnica de normalización, fíjense que lo que aparecen es el Cliente, las Ventas, los Artículos; y en este caso desglosamos Clientes y Transportes. Fíjese que esto, que si bien es un modelo de base de datos aplicado a este conjunto de datos inicial, lo que nos permite obtener es algo que se parece mucho a lo que uno termina produciendo cuando hace modelado de dominio, cuando hace modelado conceptual.

Data Mapper

Dicho esto hasta acá, vamos por el lado de base de datos relacionales porque es lo que generalmente ustedes van a usar para el trabajo práctico. Hay base de datos no estructuradas, bases de datos orientadas a documentos, etc. que utilizan un enfoque distinto. No lo vamos a ver acá pero ténganlo en el radar que no es lo único que hay. Esto se los explicamos porque es lo más fácil, lo más sencillo, y es lo que se van a encontrar con cualquiera de los frameworks que se utilizan hoy para desarrollar software. A veces, los frameworks tipo Ruby on Rails y ese tipo de cosas que van a un esquema Model View Controller, la capa de datos nos la resuelve como una base de datos y "mágicamente" nos resuelve ese mapeo entre el Model, es decir, lo que está representando de alguna manera el objeto que tenía en el modelo de dominio cuando lo traslado acá, al momento de implementación, voy a tener un objeto.



En este caso, si yo tenía un objeto Persona en mi modelo de dominio, cuando hago la traducción al diseño orientado a objetos, lo que voy a tener es una clase Persona; pero para poder resolver la persistencia de esa clase voy a tener que tener atrás una base de datos. Normalmente hay algún mapeo que me permite resolver las operaciones de: agregar una instancia del objeto Persona a la base de datos (agregar una fila sería la terminología correcta), eliminar una fila, modificar una fila. Eso muchas veces está resuelto en los

frameworks para agregarles a ustedes acá una especie de capa de abstracción que les permita manipular la base de datos sin necesidad de irse al detalle de crear las tablas, crear las columnas, crear las Primary Key, etc. Muchos de los entornos resuelven esto. Pero, más allá de que el entorno resuelva y les haga la magia, hay que entender cómo funcionan estas cosas, porque forma parte del trabajo de todos los días. Ustedes, por ahí, la basecita que van a usar en la aplicación es relativamente chica, no van a tener muchas filas, pero obviamente, en la mayoría de los casos, lo que se van a encontrar con que ya hay una base de datos que va a haber que utilizar. Les van a decir cómo utilizar esa base de datos, les van a decir cuál es el diseño y no les van a dejar modificar mucho el diseño de la base de datos, salvo que tengan una gran justificación para eso. Entonces, hay que tener 2 cosas: Por un lado, los datos hay que persistirlos. Por el otro lado, no siempre la aplicación va a ser la dueña de la base de datos, sino que va a haber que compartir la base de datos con otras aplicaciones.