Grado en Ingeniería Informática

Bases de Datos

TEMA 6

El modelo E/R y su representación lógica en SQL

Docentes:

Jesús Maudes Raedo

UBUCEV
Universidad de Burgos

Índice de contenidos

I. Introducción

En este tema suponemos que el alumno está matriculado de la asignatura de *Ingeniería del Software* que se imparte en el mismo curso y semestre, y que está familiarizado con el uso del diagrama E/R a través de dicha asignatura. Es por eso que este tema lo vemos al final del semestre, para así dar tiempo a que en *Ingeniería del* Software se vean estos contenidos.

En la asignatura de *Ingeniería del Software* los contenidos están más enfocados a obtener el diagrama a partir de una especificación de requisitos software, mientras que en este tema de la asignatura de *Bases de Datos*, nos centraremos en cómo, una vez ya se tiene el diagrama, representaremos el esquema de forma relacional, utilizando SQL, y capturando en esa representación SQL el mayor número posible de restricciones representadas en dicho diagrama. Por ello, en este tema tan sólo repasaremos el diagrama E/R, y en ese repaso, a la par, iremos introduciendo el patrón SQL que conviene utilizar en cada caso.

Otra diferencia con la asignatura de *Ingeniería del Software*, es que veremos algunos aspectos avanzados del modelo E/R que normalmente no se ven en dicha asignatura por limitaciones de tiempo (i.e., relaciones de grado superior, agregaciones y relaciones ISA).

En la práctica quien hace el diagrama, suele estar ya dando respuesta a problemas que es capaz de anticipar en cuanto cómo va a ser representación lógica en SQL de ese diagrama. Por ello, los aspectos que se contemplan en ambas asignaturas están íntimamente ligados, de manera que, es de esperar que los contenidos en una asignatura ayuden a comprender y afianzar mejor los de la otra.

II. Objetivos

OBJETIVO 1: *Repasar los elementos básicos del diagrama E/R vistos en otras asignaturas.*

OBJETIVO 2: *Dar a conocer y practicar con otros elementos avanzados del diagrama E/R.*

OBJETIVO 3: *Dominar cómo dar una representación SQL a los modelos E/R mediante las restricciones que se pueden declarar en un CREATE TABLE intentando capturar en SQL tantas restricciones expresadas en el modelo E/R como sea posible.*

OBJETIVO 4: *Hacer que el alumno tome conciencia de en qué se van a transformar sus decisiones de diseño: es decir, tome conciencia de qué implementación posible hay detrás de cada diagrama. Ayudar, de esta forma, a mejorar su intuición como diseñadores.*

III. Contenidos específicos del tema

IMPORTANTE:

1. Las secciones o subsecciones marcadas con "(*)" tienen contenidos que se supone que un alumno que haya superado o la asignatura de *Ingeniería del Software* o haya seguido los contenidos del diagrama E/R impartidos en la misma, debería de conocer. Luego son contenidos que estos alumnos se podrían saltar.
2. Las secciones o subsecciones marcadas con "(+)" tienen contenidos que quedan fuera del alcance del curso, pero que en un afán de hacer los apuntes lo más completos posible, finalmente sí que se han conservado como parte del material, pensando en aquellos alumnos que por su cuenta quieran ir un poco más allá.
3. Las secciones o subsecciones NO marcadas ni con "(*)", ni con "(+)" deben de ser estudiadas por todos los alumnos:
    1. Incluso aunque hubieran superado la asignatura de *Ingeniería del Software*
    2. Incluso aunque esa subsección esté dentro de una sección marcada con "(*)" o "(+)".

1. ¿Qué es el modelo E/R?

El modelo entidad/relación o modelo E/R es un modelo de datos, consistente en un diagrama (i.e., diagrama E/R) que sirve para hacer una primera aproximación al problema a modelar mediante una base de datos. Con este modelo, sin entrar en todos los detalles (recordemos que es un modelo conceptual, y por tanto, de alto nivel de abstracción), podemos representar qué elementos de información componen el esquema, qué atributos tiene cada uno, y cómo se relacionan entre ellos. Es por tanto, un modelo para pensar, para diseñar, y para comunicarnos con otros diseñadores. En principio, no es un modelo para ser suministrado a la máquina. No obstante, existen herramientas informáticas, como son las herramientas CASE, que proveyéndolas de algo más de información, son capaces de interpretar un diagrama y generar un esquema lógico (menos abstracto), por ejemplo deduciendo las tablas SQL que representasen el mismo problema.

El modelo E/R aparece en 1976 [Chen 1976], y hoy en día, algunos autores [Connolly y Begg 2005] han sustituido la docencia en diseño de bases de datos, tradicionalmente explicada a través del diagrama E/R, por el diagrama de clases de UML [Fowler 2003], y otros lo reseñan de manera complementaria (e.g., [Elmasri y Navathe 2007], [Silberschatz et. al 2006]), pues básicamente tienen la misma potencia visual y semántica para representar un esquema conceptual. Cualquiera de las dos alternativas es válida, los conceptos son similares, y aprender a diseñar bases de datos no depende de usar un modelo u otro, más que aprender a programar no depende de utilizar un lenguaje de programación u otro.

Nosotros utilizaremos el modelo E/R por ser aún el lenguaje más extendido en la comunidad de bases de datos, si bien cada autor de bases de datos tiende a tomarse ciertas licencias en cuanto a la modificación de los símbolos del diagrama utilizando un dialecto en particular; lo cual no ocurre con UML por ser un estándar OMG[1]. Precisamente, para tomar una referencia en cuanto a la simbología a seguir, hemos adoptado la que el Ministerio de Administraciones Públicas de España utiliza en su metodología MÉTRICA versión 3 [METRICA V3 2001].

El modelo de datos lógico con el que implementemos nuestros diseños, bien con el modelo E/R, bien con un diagrama de clases UML, no tiene por qué ser única y exclusivamente un modelo relacional en SQL, si bien ese es el caso más común. Existen otros modelos de datos, como vimos en el Tema 1, que podrían implementar estos diseños

1 *Object Management Group.*

conceptuales (e.g., una base de datos CODASYL, o una base de datos orientada a objeto, o una base de datos XML, etc …).

2. Elementos básicos del modelo E/R (*)

2.1. Conjuntos de Entidades (*)

Una ***entidad*** es:

- *A "thing" which can be distinctly identified. A specific person, company, or event is an example of an entity* [Chen 1976].
- "*Una persona, lugar, cosa, concepto o suceso, real o abstracto, de interés para la empresa"* [Tsichritzis y Klug 1978].
- *"Cualquier objeto (real o abstracto), sobre el cual queremos tener información en la base de datos"* [de Miguel y Piattini 1993].

En definitiva este concepto del modelo E/R es cualquier elemento o instancia que vamos a representar en la base de datos. Por tanto, cualquier entidad del modelo E/R se va a corresponder con una fila o tupla del modelo relacional.
Al igual que las filas o tuplas del modelo relacional se agrupan en conjuntos de tuplas homogéneos que son las tablas o relaciones, las entidades se agrupan en conjuntos homogéneos que llamaremos ***conjuntos de entidades***. Por ejemplo, el alumno *Pepe* es una entidad, pero los *alumnos* son un conjunto de entidades.
Los conjuntos de entidades se representan en el diagrama E/R a través de un rectángulo. Por ejemplo:

Figura 1: Ejemplo de Conjunto de Entidades Alumnos.

Las entidades se reconocen en una especificación textual de requisitos de la base de datos porque suelen ser sujetos o complementos (no verbos) de las frases. Por ejemplo, en un problema donde nos digan: "se quiere informatizar **una** universidad que tiene alumnos, profesores, centros y titulaciones, y las titulaciones tienen asignaturas", si nos fijamos en los sustantivos tenemos las entidades que van a salir: *alumnos, profesores, centros, titulaciones, asignaturas*. Un error de principiante es pensar que *universidad* es un conjunto de entidades, pero si lo observas con detenimiento, nos hablan de *una universidad*, y por tanto, sería un conjunto que siempre tendría un único elemento, que es esa universidad que queremos modelar. Por tanto, "en general" no tiene sentido hacer un conjunto de entidades, y a la postre una tabla, para almacenar ese único elemento. Hubiera sido distinto que nos hubieran dicho "*se quiere informatizar una red de universidades...*" ya serían varias, y merecería la pena hablar de un conjunto de entidades universidad, o de una tabla de universidades. Por tanto, un conjunto de entidades ha de indicar cierta pluralidad.
En un abuso del lenguaje, y para abreviar, la gente no habla normalmente de conjuntos de entidades, sino de entidades. Por ejemplo: *la entidad alumnos*, cuando lo estrictamente correcto es decir *el conjunto de entidades alumnos*. Nosotros, como todo el mundo, también haremos este abuso del lenguaje de ahora en adelante.
Como restricción inherente, toda entidad tiene un nombre que es único en el diagrama.

2.2. Atributos (*)

Como las relaciones en el modelo relacional, los conjuntos de entidades también tienen atributos, que serían propiedades y características comunes a las entidades pertenecientes a un mismo conjunto de entidades.

Los atributos del diagrama E/R se representan a través de elipses unidas por arcos al rectángulo correspondiente a la entidad a la que pertenecen. Por ejemplo:

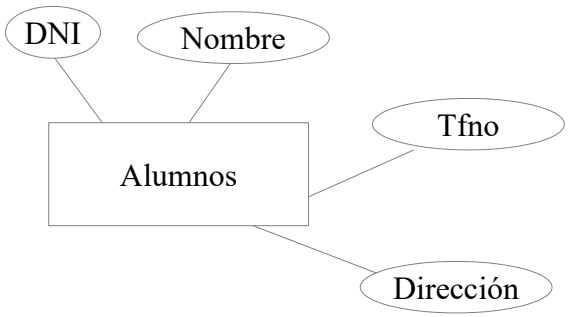


Figura 2: Ejemplo de Entidad con sus Atributos.

Como restricción inherente, nuevamente un atributo tiene un nombre único en cada entidad. Como restricciones semánticas, en el diagrama E/R podemos definir:

1. ***AIP (Atributo Identificador Principal)***, [de Miguel y Piattini 1993], que es un atributo que sirve para identificar una entidad dentro de un conjunto de entidades. La restricción equivalente en el modelo relacional, es la clave primaria. Un conjunto de entidades sólo puede tener un AIP. En principio asumiremos que toda entidad tiene una AIP, aunque más tarde veremos excepciones a esta restricción.
2. ***AIA (Atributo Identificador Alternativo)***, [de Miguel y Piattini 1993], que es un atributo cuyos valores no se van a repetir en las entidades de un conjunto de entidades. La restricción equivalente en el modelo relacional, es la clave candidata no primaria (UNQ según nuestra notación). Un conjunto de entidades puede tener o no AIA, o puede tener también varios.

Los AIPs se representan subrayando el atributo en cuestión. En el ejemplo anterior, suponiendo que el DNI fuese AIP:

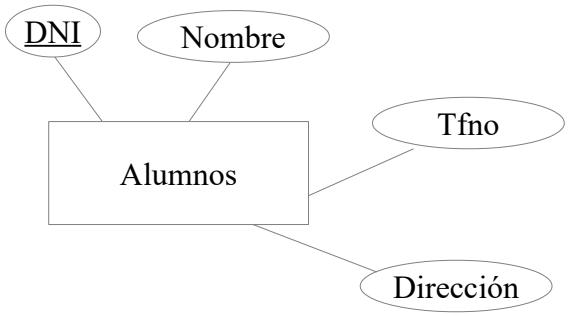


Figura 3: Ejemplo de Entidad con sus Atributos y AIP simple.

Los AIAs no tienen en principio una representación específica aunque si que hay notaciones que lo permiten [de Miguel y Piattini 1993], como la del siguiente ejemplo, suponiendo que *DNI* sea un AIA e *IdAlumno* un AIP:

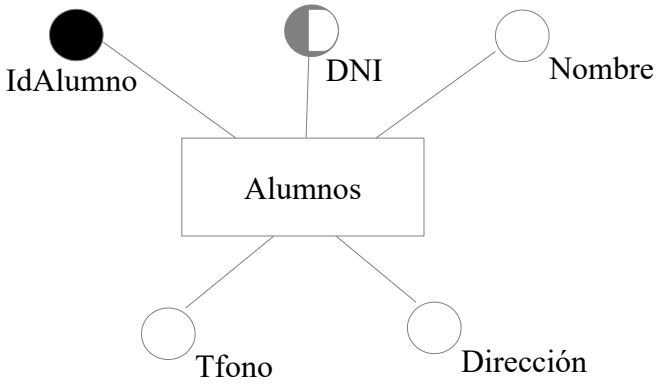
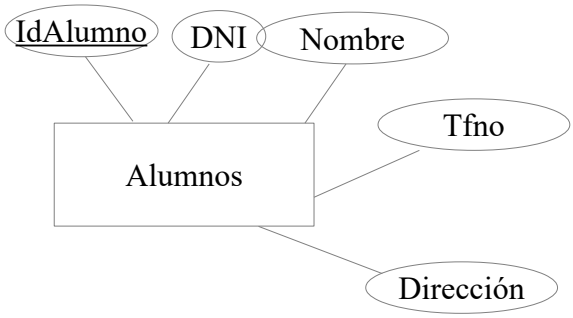


Figura 4: E/R alternativo que permite representar AIAs.

En esta notación los atributos son puntos, que están rellenos para los AIPs y semirrellenos para los AIAs. Esta notación no está muy extendida y nosotros preferiremos la primera. Como el diagrama E/R puede ser completado por una lista de aclaraciones, podríamos indicar en dichas aclaraciones quiénes son los AIAs. Por ejemplo:



- El atributo DNI de Alumnos es AIA

Figura 5: Ejemplo de Entidad con sus Atributos y AIP simple.

En el modelo E/R puede haber atributos compuestos[2]. Un atributo compuesto se representa como un racimo. Supongamos que un cliente tiene un atributo "*número de cuenta*" al que le cargamos las facturas de las compras que nos hace. Consideramos que la cuenta es un atributo compuesto de los siguientes atributos simples: *númeroBanco, númeroSucursal, dígitoControl* y *númeroCta*. En el modelo E/R podemos representarlo así:

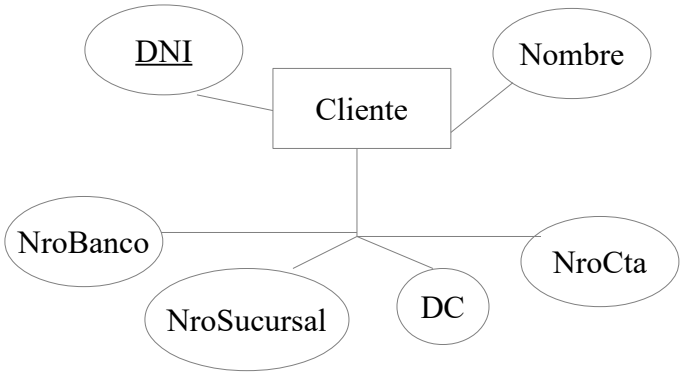
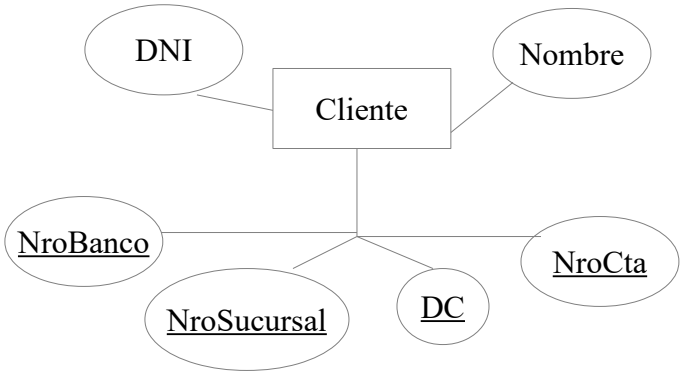


Figura 6: Ejemplo de Entidad con un Atributo Compuesto.

Al poder representar atributos compuestos, podemos representar AIPs y AIAs compuestos de la misma manera. Por ejemplo, supongamos que el atributo compuesto de número de cuenta ahora es AIP, y DNI queda como AIA, los diagramas resultantes serían los siguientes:

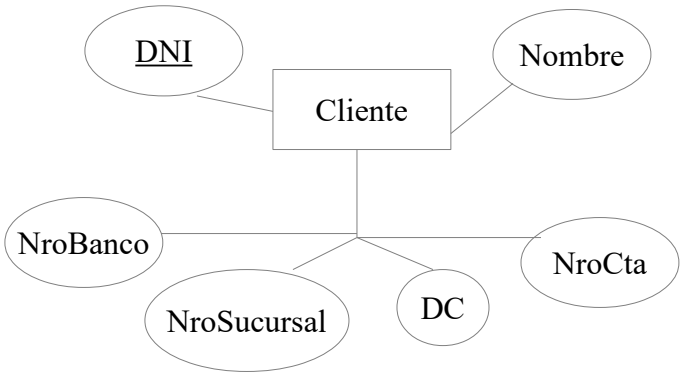
2 Esta es una diferencia con el modelo relacional, recuerda que un atributo relacional era "*atómico*" para preservar la primera forma normal.



- DNI de Clientes es AIA

Figura 7: Ejemplo de Entidad con un AIP Compuesto.

Si la cuenta corriente fuera AIA y el DNI AIP:



- NroBanco+NroSucursal+DC+NroCta de Clientes es AIA

Figura 8: Ejemplo de Entidad con un AIA Compuesto.

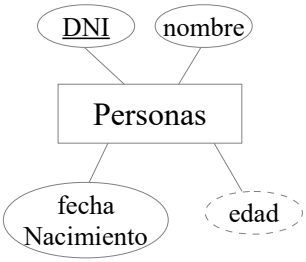
2.3. Paso al Modelo Relacional de Entidades Fuertes y Atributos monovaluados

Cada entidad fuerte genera una tabla. Todos los atributos simples de la entidad pasan a ser atributos de la tabla. Si fuesen compuestos, se genera un atributo de la tabla por cada atributo componente. El AIP o clave pasa a ser clave primaria de la tabla, y los AIAs pasan a ser claves candidatas no primarias (UNIQUEs). En principio todos los UNIQUEs que declaremos serán NOT NULL.

2.4. Atributos derivados (*)

En el modelo E/R es posible representar un tipo especial de atributos, llamados atributos derivados, cuyos valores se pueden calcular/derivar a partir de valores de otros atributos de la base de datos.

Por ejemplo, supongamos el siguiente diagrama E/R:



- Personas.Edad = Parte Entera(Fecha de hoy - Personas.Fecha de Nacimiento)/365)

Figura 9: Atributo derivado edad.

La edad se puede calcular a través de la fecha de nacimiento, por ello es adecuado pensar que se trata de un atributo derivado. Lo representamos con una elipse en línea discontinua.
La regla de cálculo correspondiente a un atributo derivado puede adjuntarse en los comentarios del diagrama, como ocurre en la figura.

2.4.1.1 Paso a tablas de los atributos derivados (+)

Los atributos derivados, en principio no les daremos ninguna representación. Si acaso, existen algunas posibilidades que no exploraremos este curso, a saber:

1. Crear una vista sobre la tabla que genera la entidad y crear el campo calculado mediante una expresión proyectada por la SELECT en la que se basa la vista.
   Por ejemplo, para la Figura 9:

```
CREATE TABLE personas (
 DNI                NUMERIC(8)     PRIMARY KEY,
 nombre             VARCHAR(40),
 fechaNacimiento    DATE);
CREATE VIEW v_personas AS
 SELECT *, (CURRENT_DATE- fechaNacimiento)/365 as edad
 FROM personas;

/*Al ser una división entre 2 enteros, ya quita los decimales
redondeando hacia abajo sin necesidad de hacer más*/
```

2. Crear la tabla que genera la entidad incluyendo ese campo calculado como si fuera otro campo mas. A continuación, crear un sistema de varios disparadores *triggers* que impidan actualizar ese campo y que lo actualicen automáticamente a partir de los campos que actúan como operandos en su regla de cálculo. (Esta aproximación se ve en la asignatura de *Aplicaciones de Bases de Datos* de tercer curso).
3. El estándar de 2003 definió un nuevo tipo de campos llamados *Generated Columns* [Eisenberg et al. 2004], que permiten representar de una forma sencilla los campos calculados en el propio comando CREATE TABLE. Desafortunadamente pocos sistemas lo tienen (e.g., Oracle a partir de la versión 11[3]).

3 http://www.oracle.com/technetwork/articles/sql/11g-schemamanagement-089869.html (ultima visita 22-08-2015)

```
CREATE TABLE personas (
 DNI                NUMERIC(8)      PRIMARY KEY,
 nombre             VARCHAR(40),
 fechaNacimiento    DATE,
 edad               GENERATED ALWAYS AS
          (CURRENT_DATE- fechaNacimiento)/365 );
```

3. Interrelaciones o Relaciones (*)

Hasta ahora, las entidades aparecían solas e inconexas en el diagrama E/R. Por ejemplo: supongamos que el enunciado de un problema que nos pide hallar el diagrama E/R nos dice: *"se quiere informatizar un hospital, y para ello necesitamos la información de los **pacientes**, los cuales tienen una **habitación** cada uno que puede ser compartida, …"*. Con nuestros conocimientos actuales a lo sumo podríamos hacer la siguiente representación:

Figura 10: Representación de dos conjuntos de entidades no relacionados en el diagrama E/R.

Sin embargo, sabemos que a un paciente le corresponde una única habitación, y que en una habitación puede que haya varios pacientes. Desde nuestro conocimiento del modelo relacional, sabemos que las tablas de *Pacientes* y *Habitaciones* que resulten habrán de estar relacionadas por una clave ajena, probablemente el número de la habitación. ¿Podemos representar esto en el modelo E/R visualmente?. La respuesta es sí, pero para ello necesitamos ampliar los conceptos del modelo E/R.

Una ***relacion*** o ***interrelación***[4] es una asociación de varias entidades (entendidas como instancias, no como conjuntos de entidades). Por ejemplo, entre el paciente *Pepe* y la *habitación 14* existe una interrelación, pues ese paciente está asociado a dicha habitación.

El ***grado*** de una interrelación es el número de entidades que asocia. Por lo tanto el grado de la interrelación entre una habitación y un paciente es dos, y se dice que es binaria. Existen interrelaciones de grado superior a dos, como veremos al final del tema.

Si juntamos en un conjunto todas las interrelaciones que:

1. Asocian elementos de los mismos conjuntos de entidades, y
2. Significan lo mismo,

obtendremos un ***conjunto de interrelaciones***. Por ejemplo, en el conjunto de interrelaciones "*estar ingresado en*" siempre se asocia un elemento del conjunto de pacientes con el de habitaciones, y el significado de esa asociación es siempre el mismo: que ese paciente está ingresado en esa habitación.

4 En la mayoría de textos se habla de ***relaciones***, en lugar de ***interrelaciones*** [de Miguel y Piattini 1993]. En el fondo es una cuestión de cómo traducir al castellano la palabra inglesa *relationship*. La preferencia por traducirla como interrelación puede estar justificada para evitar confusiones con el concepto/palabra *relación* propio del modelo relacional. En inglés no existe esta confusión, porque *relación* del modelo relacional se dice *relation* y no *relationship*.

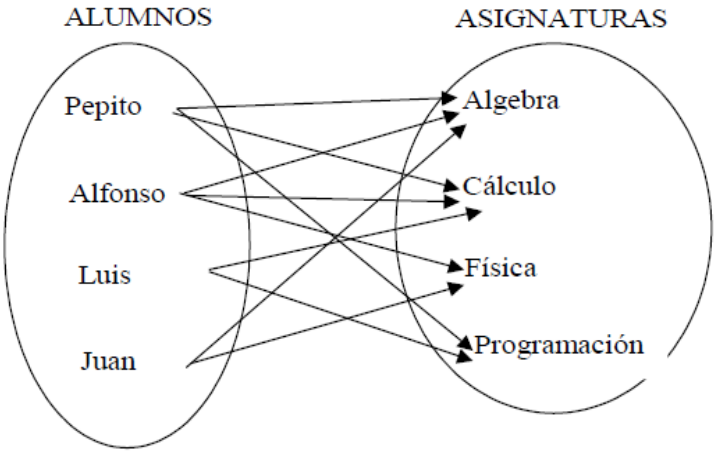


Figura 11: Representación vía diagrama de relaciones de un *conjunto de interrelaciones*.

En la figura vemos la representación de la interrelación "*estar matriculado de*" entre *alumnos* y *asignaturas.* Cada elemento del conjunto de interrelaciones es una de las flechas, o si queremos también, los pares { *(Pepito, Álgebra), (Pepito, Cálculo), (Pepito, Programación), (Alfonso, Álgebra), (Alfonso, Cálculo), (Alfonso, Física), (Luis, Cálculo), (Luis, Programación), (Juan, Álgebra), (Juan, Física)* }. La dirección de la flecha no nos importa.

El conjunto de interrelaciones, obviamente es un "conjunto" en el sentido matemático, y por lo tanto no tiene repetidos: cada una de las flechas es única, y está identificada por los extremos o entidades que une. Por ejemplo, la flecha (*Juan, Física*). Cada elemento de un extremo sólo puede estar relacionado una vez con el elemento del otro extremo[5].

Las interrelaciones de un conjunto de interrelaciones tienen todas el mismo grado, y por ello hablamos de ***grado del conjunto de interrelaciones***.

Podría haber otros conjuntos de interrelaciones definidos entre los mismos conjuntos de entidades, pero entonces seguramente tendrán distinto significado. Por ejemplo, entre dos conjuntos de entidades: *profesores* y *departamentos* podría haber dos conjuntos de interrelaciones: (I) "*pertenecer al departamento*", (II) "*ser director del departamento*". Ambas relacionan elementos de los mismos conjuntos de entidades, pero aún cuando casualmente coincidiese en que los departamentos fueran todos de un único profesor, el cual fuese asimismo director (i.e., las flechas de ambos conjuntos de interrelaciones fuese el mismo), el significado de ambos conjuntos de interrelaciones es distinto.

Como en el caso de los conjuntos de entidades, nadie habla de conjuntos de interrelaciones; es demasiado largo de decir, y casi todo el mundo prefiere hablar de *interrelaciones* (*relaciones*) a secas. Eres tú a través del contexto, el que tiene que distinguir si te quieren decir *conjunto de interrelaciones* o *interrelaciones.*

Los conjuntos de interrelaciones tienen una representación diagramática según el modelo E/R, consistente en un rombo del que parten tantos arcos como grado tenga la interrelación, cada arco llega a uno de los rectángulos correspondientes a las entidades que asocia. Por ejemplo, en el caso de los *pacientes* y *habitaciones* sería así:

Figura 12: El rombo representa un conjunto de interrelaciones.

5 Si que parece lógico que un alumno sólo esté matriculado una vez de una asignatura, pero quizás estés pensando en qué pasa cuando ese alumno se ha matriculado en varias ocasiones porque es repetidor. Ya analizaremos ese caso más adelante, cuando veamos las interrelaciones de grado superior a dos.

¿Cómo reconocer las interrelaciones cuando estemos modelando un problema?. No hay una regla infalible, pero un buen consejo es fijarnos en los verbos del enunciado. Por ejemplo, si nos dicen "se quiere informatizar un hospital en el que hay pacientes y habitaciones. Los pacientes están ingresados en habitaciones...", la forma verbal *estar ingresado* que toma como sujeto la entidad *paciente*, y como complemento *habitaciones*, lingüísticamente hace el mismo papel que el rombo en el diagrama: expresar que los elementos de un conjunto se relacionan con los del otro.

4. Cardinalidades de las interrelaciones binarias (*)

Como cualquier modelo de datos, el modelo E/R posee restricciones semánticas que permiten modelar el problema. Una de estas restricciones son las restricciones de cardinalidad de las interrelaciones.

Una interrelación binaria (entendida como conjunto) tendrá una restricción de cardinalidad por cada una de las entidades que conecta. La restricción de cardinalidad figurará en cada arco de la interrelación. Cada una de estas restricciones, a su vez es un par de números: la cardinalidad mínima y la máxima.

Sea *R* una interrelación que relaciona las entidades *E1* y *E2.* La restricción de cardinalidad en el arco que conecta el rombo de la con *E1* indica con cuántos elementos de *E1* se relaciona *E2.* Por ejemplo, si *Están Ingresados* es una interrelación entre las entidades *Pacientes* y *Habitaciones,* la cardinalidad en el lado de pacientes indica cuántos pacientes hay en la habitación, y la del lado de habitaciones indica cuántas habitaciones ocupa un paciente.

Figura 13: Etiquetas representando las cardinalidades.

La cardinalidad puede ser mínima o máxima: la mínima indica si una instancia de entidad puede existir sin participar en la interrelación, y puede valer 0 ó 1. Cuando vale 0 indica que sí puede existir sin participar en la interrelación, y cuando vale 1, indica que no puede existir sin participar en la interrelación. Por ejemplo, si la cardinalidad mínima del lado *habitación* en la relación *estar ingresado* fuese 0, significaría que puede haber pacientes que no participen en esa interrelación y que por lo tanto no tengan habitación asignada. Pero si fuese uno significaría que en nuestro modelo de hospital no se puede ser paciente sin tener asignada una habitación. Como vemos la situación es discutible, dependerá de cómo quiera la gerencia del hospital que modelemos el funcionamiento del mismo.

Analicemos la cardinalidad mínima del lado *pacientes* en la misma interrelación. Claramente es 0 siempre, pues una habitación puede existir sin estar relacionada con ningún paciente (la habitación estaría vacía).

La cardinalidad máxima, sin embargo, indica si una instancia de entidad puede participar varias veces o tan sólo una en una interrelación. En el ejemplo, en el lado de *pacientes* valdrá "*n"*, y en el otro valdrá 1.

Por ejemplo, si una habitación del hospital sólo pudiera albergar a un paciente, la cardinalidad máxima del lado *pacientes* sería 1. Pero si las habitaciones tuvieran varias camas sería "*n"*, como ocurre en la figura. Por otro lado, el paciente sólo puede estar en una habitación, luego parece claro que la cardinalidad máxima de lado *habitaciones* en esa interrelación es 1.

Las ***interrelaciones binarias posibles*** según su cardinalidades máximas son:

1. **Uno a Uno** (también conocida como ***1:1***), una entidad del conjunto origen está asociada a lo sumo a una del imagen y viceversa. Por ejemplo, la interrelación "*ser director de*" entre profesores y departamentos
2. **Uno a Varios** (también se dice uno a muchos o ***1:N***), una entidad del conjunto origen puede estar asociada a varias entidades del conjunto destino, pero una entidad del conjunto destino sólo puede estar asociada a lo sumo con una entidad del origen. El caso *Varios a Uno*, es el mismo. Por ejemplo, "*pertenecer a*" entre

profesores y *departamentos",* puede considerarse 1:N o N:1 con sólo cambiar las posiciones de las entidades en el dibujo.

En las interrelaciones uno a varios, a veces hablaremos de la **parte "a uno"** y de la **parte "a varios"**. La parte "a uno" es la entidad etiquetada con la cardinalidad máxima "1". Por ejemplo, *habitación* en la Figura 13. Por el contrario la parte "a varios" es la entidad etiquetada con la cardinalidad máxima "n". Por ejemplo, *pacientes* en la Figura 13.

3. **Varios a Varios** (también se dice muchos a muchos o ***M:N***), una entidad del conjunto origen puede estar asociada a varias entidades del conjunto destino y viceversa. Por ejemplo, "*impartir*" entre profesores y asignaturas.

A continuación analizaremos estos primeros casos.

4.1. Cardinalidades uno a varios (*)

Habría cuatro situaciones producto de combinar que en un extremo haya (0, n) o (1, n), y que en el otro haya (0, 1) o (1, 1):

Figura 14: Ejemplo (0,1) – (0,n).

- (0, 1) y (0, n), ya hemos analizado el ejemplo de los *pacientes* y las *habitaciones*. Los pacientes pueden no tener habitación y si la tienen sólo tienen una (0, 1). Las habitaciones podrían estar vacías, pero pueden tener varios pacientes (0, n).

Figura 15: Ejemplo (0,1) – (0,n).

- (1, 1) y (0, n), por ejemplo, un recluso tiene que tener obligatoriamente una celda, y sólo una (1, 1), pero pudiera ser que la celda estuviese vacía o fuese compartida (0, n).

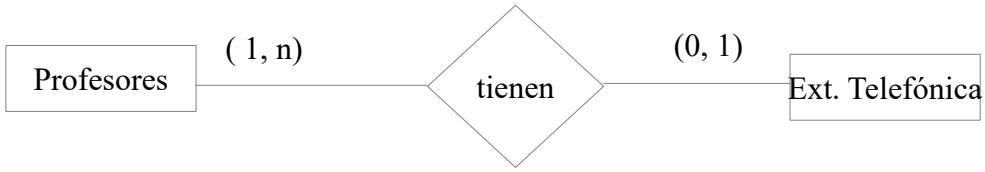


Figura 16: Ejemplo (0,1) – (1, n).

- (0, 1) y (1, n). Por ejemplo, supongamos que no podemos poner líneas de teléfonos que no corresponda a algún profesor, pero que la misma extensión de teléfono puede ser compartida por varios profesores (1, n). Supongamos además que un profesor no puede tener más de una línea telefónica, y que pudiera no tener ninguna (0, 1).

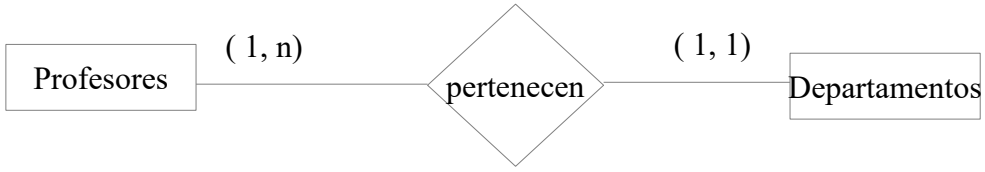


Figura 17: Ejemplo (1, 1) - (1, n).

- (1, 1) y (1, n). Por ejemplo, un profesor de la universidad siempre ha de pertenecer a un departamento, y sólo a uno (1, 1), pero en un departamento podría haber varios profesores, y al menos debería de haber uno (1, n).

4.2. Paso al Modelo Relacional de las Interrelaciones 1:N

Existen unas reglas que nos permiten pasar a tablas relaciones, las interrelaciones que hayamos plasmado en el diagrama E/R. En este punto nos centraremos en las interrelaciones 1:n:

1. Por cada interrelación binaria 1:N, añadiremos los atributos de la clave primaria de la tabla correspondiente a la entidad con el papel de parte "a uno", a la tabla correspondiente a la entidad con el papel "a varios". Declararemos una clave ajena en la tabla de la parte "a varios" compuesta por esos atributos y apuntando a la tabla de la parte "a uno".
2. Si la cardinalidad mínima de la parte "a uno" es 1 (si cada entidad de la parte a varios se relaciona como mínimo con una de la parte a uno), los atributos de la clave ajena serán NOT NULL.

Observa que en el caso 1:N la cardinalidad mínima en el lado "a varios" nos da igual que valga 0 ó 1; porque el caso (1, n) no es controlable desde restricciones declarables en un CREATE TABLE, habría que recurrir a implementaciones más avanzadas de las restricciones (e.g., ASSERTIONs[6], *triggers* que ya comentaremos más adelante en la sección 6.2.2) que se quedan fuera del alcance del curso y que pueden ralentizar a la base de datos. Por ejemplo, en el diagrama:

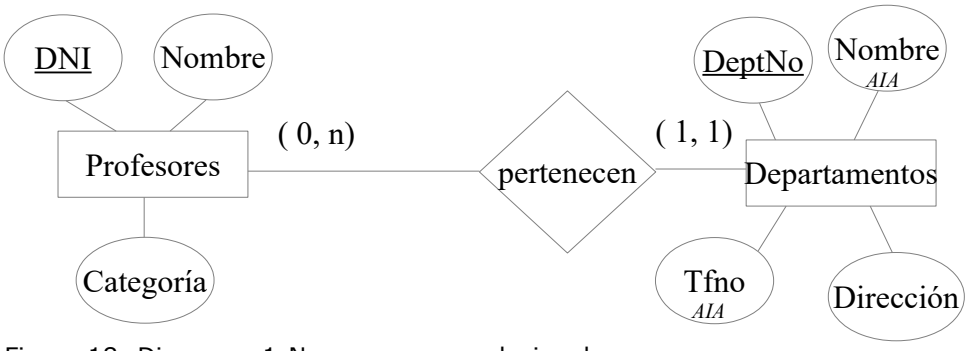


Figura 18: Diagrama 1:N para pasar a relacional.

La parte "a varios" es profesores, luego la clave primaria de departamentos pasaría a ser clave ajena en profesores quedando una declaración SQL como la siguiente:

```
CREATE TABLE Departamentos (
 Deptno    INTEGER    PRIMARY KEY,
 Nombre    CHAR(40)   UNIQUE NOT NULL,
 Tfno                 NUMERIC(9) UNIQUE NOT NULL,
 Dirección            CHAR(60)
);
```

6 En el Glosario parece una explicación detalla de lo que es uno objeto ASSERTION, con el correspondiente ejemplo.

```
CREATE TABLE Profesores (
 DNI       CHAR(9)    PRIMARY KEY,
 Nombre          CHAR(40)  NOT NULL,
 Categoría       CHAR(4)    NOT NULL,
 Deptno          INTEGER NOT NULL REFERENCES
                                        Departamentos
);
```

El NOT NULL en *Deptno* de *Profesores* viene justificado porque la cardinalidad mínima de la parte "a varios" es uno. Es decir, a cada profesor le corresponde obligatoriamente un departamento. Si el (1, 1) del diagrama le sustituimos por (0, 1) no habría que poner el NOT NULL en dicho campo.

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

Si la clave primaria de departamentos hubiese sido compuesta hubiéramos procedido de la misma forma. Por ejemplo, supongamos que el departamento se codifica a partir del número de centro donde se ubica (*CentroNo*), que por ejemplo puede valer 1 si es derecho, 2 si es económicas, 3 si es la politécnica etc.. y *DeptNo* es un número relativo a cada centro, de forma que puede haber varios departamentos con el mismo *Deptno*, pero tendrían distinto *CentroNo.*

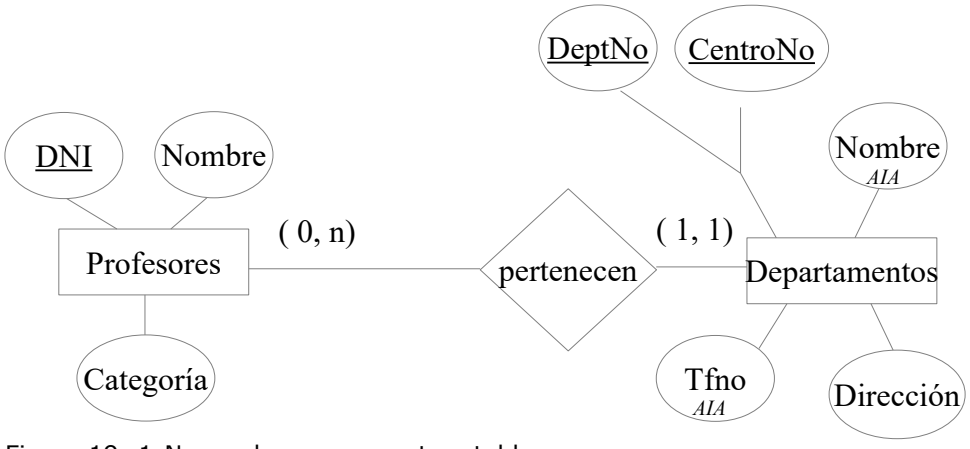


Figura 19: 1:N con clave compuesta a tablas.

Eso daría lugar al siguiente esquema relacional en SQL:

```
CREATE TABLE Departamentos (
     Deptno          INTEGER,
     CentroNo        INTEGER,
     Nombre          CHAR(40)        UNIQUE NOT NULL,
     Tfno            NUMERIC(9)      UNIQUE,
     Dirección       CHAR(60),
     PRIMARY KEY (CentroNo, DeptNo)
);
```

```
CREATE TABLE Profesores (
     DNI            CHAR(9)    PRIMARY KEY,
     Nombre         CHAR(40)  NOT NULL,
     Categoría      CHAR(4)    NOT NULL,
     Deptno         INTEGER    NOT NULL,
     CentroNo       INTEGER    NOT NULL,
     FOREIGN KEY (CentroNo, DeptNo) REFERENCES
                                   Departamentos
);
```

Ahora son *Deptno* y *CentroNo* de *Profesores* los campos NOT NULL por ser la cardinalidad mínima (1, 1).

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

***Ejercicio**: Haz un diagrama E/R genérico correspondiente a dos entidades interrelacionadas por una relación uno a varios. Utiliza letras para designar los nombres de las 2 entidades, la relación y los atributos que necesites. Deduce las dependencias funcionales y las claves. ¿A que forma normal llegas y por qué?.*

4.2.1 Caso en el que sean muy pocos los elementos del lado "a varios" que participan en la interrelación (+)

En [de Miguel y Piattini 1993] pgs 754-755, se discute este caso, que es poco usual. En esta situación el porcentaje de valores nulos en la clave ajena que generaría la interrelación 1:N sería muy alto, y excepcionalmente podría estar permitido tratar la interrelación 1:N de forma parecida como veremos más adelante se pasa a tablas a una M:N. Esto significa:

1. Crear las dos tablas correspondientes a las dos entidades de la interrelación tal cual (i.e., sin la clave ajena).
2. Crear una tabla intermedia cuyos campos sean los AIPs de las dos entidades, ambos AIPs serán claves ajenas a cada una de las entidades, y
3. (aquí radicaría la diferencia con el caso N:M) la clave primaria de la nueva tabla sería el AIP de la parte "a varios".

4.3. Cardinalidades uno a uno (*)

Las cardinalidades máximas uno a uno pueden dar lugar a tres tipos de interrelaciones dependiendo de cuáles sena las cardinalidades mínimas en cada uno de los extremos:

- (1, 1) en ambos extremos, por ejemplo supón que tenemos una entidad *ordenadores* y otra entidad *teleoperadores*, de forma que en nuestra base de datos, cada teleoperador siempre tiene un ordenador y uno sólo, y cada ordenador se corresponde siempre con un teleoperador, y sólo uno.

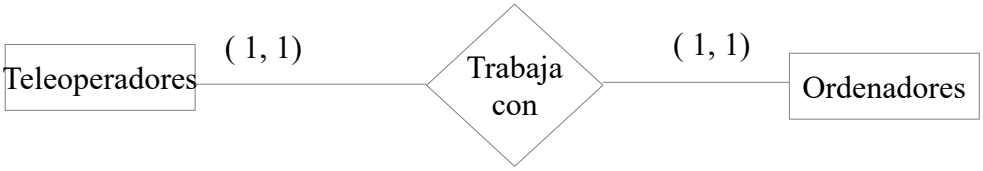


Figura 20: Ejemplo (1, 1) - (1, 1).

- (1, 1) en un lado y (0, 1) en el otro, por ejemplo es el caso de los *departamentos* y los *profesores*, en el caso de la relación "*dirige*".

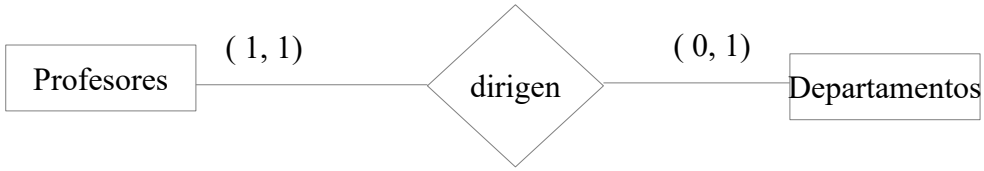


Figura 21: Ejemplo (1, 1) ⟷(0, 1).

Departamento siempre tiene un director y sólo uno, por ello en la izquierda hay un (1, 1). Pero un profesor puede que no sea director del departamento (el cero), y si lo es sólo es director de uno, por ello en la derecha hay (0, 1).

- (0, 1) en los dos lados, por ejemplo la interrelación "*trabaja con*" del ejemplo (1, 1) ↔ (1,1) que acabamos de ver, podría ser (0, 1) ↔ (0,1) si cambiamos los supuestos del enunciado. Por ejemplo, cada ordenador si está asignado a algún teleoperador, está asignado como mucho a uno, pero hay ordenadores no asignados a nadie en la base de datos, porque por ejemplo sean servidores. En el otro lado puede que haya teleoperadores sin ordenador, quizás porque trabajan desde casa con un ordenador propio (que no es de la empresa) y la empresa no tiene registrado en su base de datos. Pero, igual que antes, si el teleoperador tuviese asignado algún ordenador de empresa, tendría a lo sumo uno.

4.4. Paso al Modelo Relacional de las Interrelaciones 1:1

Existen 3 posibilidades :

4.4.1 En ambos lados las cardinalidades son (1, 1) (+)

Este caso, se discutió en la sección 4.5.3.1 y puede encontrarse en [Connolly y Begg 2005] sección 16.1(4)(b). En ese caso, la representación es de una única tabla conteniendo la información de ambas entidades.

Supongamos el siguiente ejemplo, y el correspondiente CREATE TABLE.

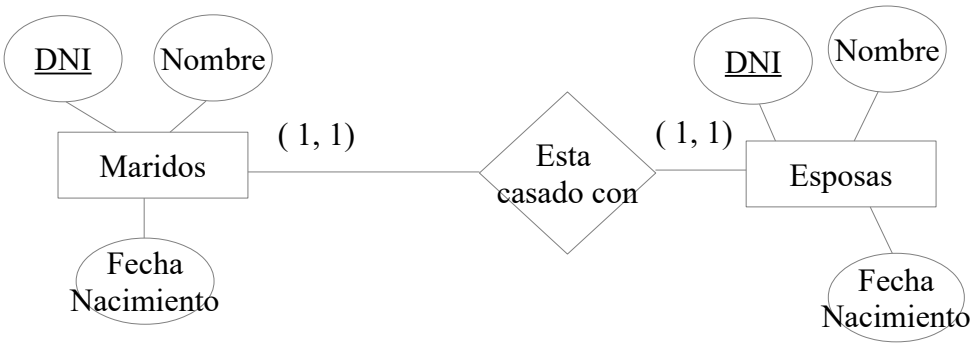


Figura 22: Diagrama 1:1 con 2 cardinalidades mínimas 1.

```
CREATE TABLE matrimonio(
     DNIconyuge1          NUMERIC(8) PRIMARY KEY,
     DNIconyuge2          NUMERIC(8) UNIQUE NOT NULL,
     nombreConyuge1       VARCHAR(40) NOT NULL,
     nombreConyuge2       VARCHAR(40) NOT NULL,
     fechaNacConyuge1     DATE,
     fechaNacConyuge2     DATE,
     CHECK (DNIconyuge1 > DNIconyuge2) );
```

Nota: El CHECK final sirve para prevenir por un lado: que alguien se case consigo mismo, y por otro: para tener un criterio "no – sexista" a la hora de elegir qué DNI de entre los 2 posibles identificaría el matrimonio, aunque siempre hubieran quedado otras posibilidades (e.g., el más viejo de los dos).

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

4.4.2 En algún lado la cardinalidad es (0,1) y gran parte de los elementos participan en la interrelación

Este es el caso más común entre las interrelaciones 1:1. Lo trataremos como un caso especial de interrelación 1:N en el que haremos que la clave ajena sea además UNIQUE (i.e., clave candidata); concretamente:

1. Si un extremo es (0,1) y el otro (1,1) declararemos la clave ajena en la parte (0,1) y haremos que esa clave sea UNIQUE y NOT NULL.
2. Si los dos extremos son de igual cardinalidad mínima, nos dará igual la dirección en que se propague la clave, "en principio" elegiremos a nuestro libre criterio cual de las dos tablas se la queda. La clave ajena será UNIQUE, pero **no** NOT NULL.

Veamos un caso de las relaciones 1:1, en el que la cardinalidad mínima es distinta en cada extremo.

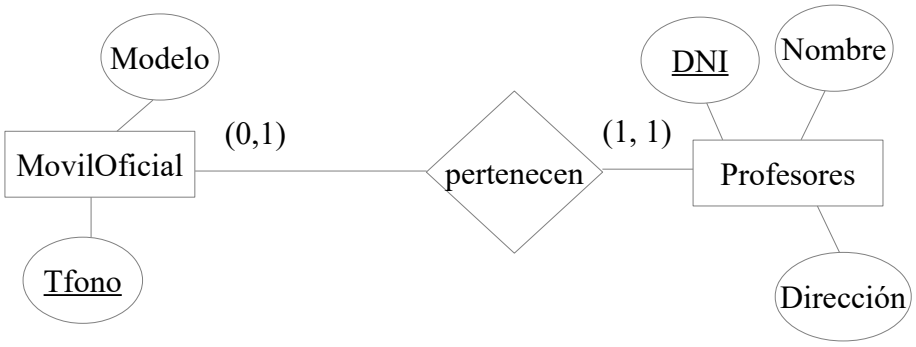


Figura 23: Diagrama 1:1 asimétrico para pasar a relacional.

Supongamos que algunos de los profesores de la universidad tienen un teléfono móvil de la universidad, y que cada teléfono está asignado siempre a algún profesor. Esto genera un esquema SQL como el siguiente:

```
CREATE TABLE profesores (
 DNI CHAR(9)    PRIMARY KEY,
 Nombre         CHAR(40) NOT NULL,
 Dirección      CHAR(60)
);

CREATE TABLE movilOficial (
 Tfno               NUMERIC(9)      PRIMARY KEY,
 DNIProfe CHAR(9)   NOT NULL UNIQUE REFERENCES
                                    profesores,
 modelo             CHAR(10)
);
```

¿Por qué hemos elegido que la clave ajena vaya en la tabla de móviles?. Porque queremos un esquema SQL en el que podamos restringir que haya móviles sin profesor (fijémonos en que la cardinalidad minima "1" que hace que un móvil tenga que pertenecer al menos a un profesor), y haciéndolo de esta manera lo hemos conseguido, porque tenemos un *DNIProfe* en la tabla de móviles que hemos podido declarar como NOT NULL.

Si lo hubiéramos hecho al revés, en la tabla de profesores tendríamos que poner un campo móvil oficial como clave ajena, pero la tabla de móviles ya no tendría el campo con el DNI del profesor, y ya no podríamos prohibir móviles sin profesor.

El UNIQUE para *DNIprofe* controla que un DNI de un profesor no se pueda repetir en la tabla de móviles, evitando así que un profesor tuviese más de un móvil. Así estaríamos implementando la cardinalidad máxima 1 del lado *MovilOficial*.

Finalmente, la cardinalidad máxima 1 del lado de Profesores nos indica que un móvil no puede ser de más de un profesor, pero esta restricción está también cubierta porque para violarla el mismo teléfono debiera aparecer más de una vez en la tabla de móviles (una por cada DNI de profesor que lo comparte). Pero si fuese así se violaría la clave primaria de móviles oficiales.

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

Ahora veamos un caso en el que ambos extremos son (0, 1). Por ejemplo:

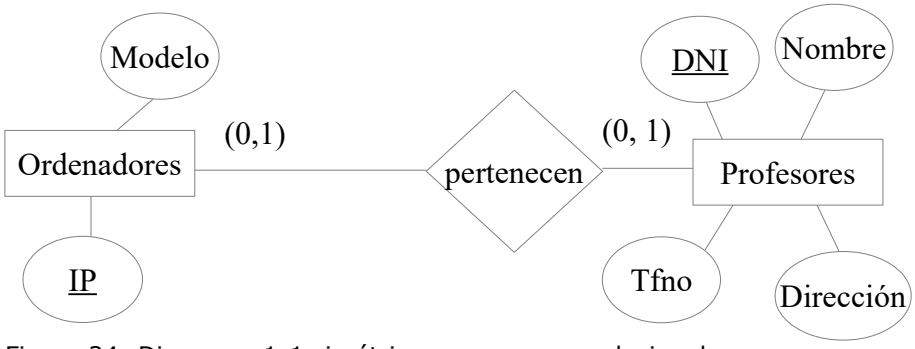


Figura 24: Diagrama 1:1 simétrico para pasar a relacional.

Supongamos que un profesor pueda no tener ordenador, por ejemplo el día que lo contratan todavía no tiene ordenador a la espera de que se compre. Supongamos que hay ordenadores no asignados a ningún profesor en concreto, por ejemplo un servidor web. Ahora bien, si un ordenador está asignado a un profesor, sólo lo está a uno, y un profesor tampoco puede tener más de un ordenador. El SQL resultante podría ser:

```
CREATE TABLE profesores (
 DNI            CHAR(9)         PRIMARY KEY,
 Nombre         CHAR(40)        NOT NULL,
 Tfno           NUMERIC(9),
 Dirección      CHAR(60)
);
```

```
CREATE TABLE ordenadores (
 IP       NUMERIC(12)    PRIMARY KEY,
 DNIProfe CHAR(9)        UNIQUE REFERENCES profesores,
 modelo   CHAR(10)
);
```

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

(En la tabla ordenadores *IP* como PK controla que un ordenador aparezca dos veces, y por lo tanto evita que sea compartido por 2 o más profesores, y *DNIProfe* como UNIQUE, evita que el mismo DNI aparezca varias veces, y por lo tanto controla que un profesor no tenga más de un ordenador).

Pero indistintamente podría haberlo hecho al revés:

```
CREATE TABLE ordenadores (
 IP       NUMERIC(12)    PRIMARY KEY,
 modelo   CHAR(10)
);

CREATE TABLE profesores (
 DNI           CHAR(9)   PRIMARY KEY,
 Nombre        CHAR(40)  NOT NULL,
 Tfno          NUMERIC(9),
 Dirección     CHAR(60),
 IP       NUMERIC(12)    UNIQUE REFERENCES ordenadores
);
```

(En la tabla profesores DNI como PK controla que un profesor no tenga 2 o más ordenadores, y el UNIQUE controla que un ordenador no sea compartido por varios profesores).

4.4.2.1 Criterio avanzado para decidir en qué lado ponemos la clave ajena en las interrelaciones 1:1 con cardinalidades (0, 1) en los dos lados (+)

Otra posibilidad para decidir cuál es la mejor opción, es ver que la relación entre profesores sin ordenador y ordenadores sin profesores no está tan equilibrada. Por ejemplo, ¿qué es más normal¿, ¿que haya profesores sin ordenador u ordenadores sin profesor?. Las dos situaciones anteriores provocarán valores nulos en cualquiera de las soluciones, y el minimizar el numero de nulos que haya en las filas de nuestras tablas, siempre es buena cosa.

Por ejemplo:

1. Si nos dicen que casi todos los profesores tienen ordenador (de manera que es más normal que un ordenador no tenga profesor, mientras que es menos normal que un profesor no tenga ordenador asignado), podemos pensar que es un caso que esta cerca de:

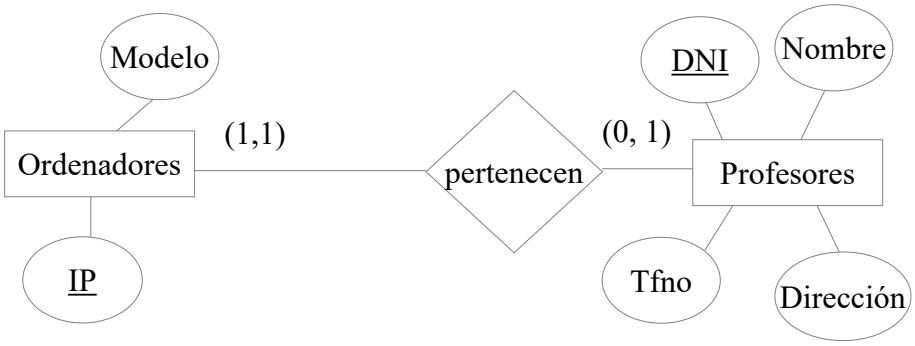


Figura 25: Imaginando un caso (0,1) - (0,1) cercano al (1, 1) - (0, 1).

Y por tanto optaríamos por la solución de ese caso, es decir:

- Profesores añade un campo IP que es clave ajena y UNIQUE (aunque no es NOT NULL, porque aunque improbable, sigue siendo posible tener profesores sin ordenador).
- Este campo sólo vale NULL en los pocos casos en los que el profesor no tenga ordenador, por lo que casi nunca es nulo. Esto es lo que hace a la solución ventajosa, pues hemos minimizado el número de posibles nulos.

2. Si nos dicen que casi todos los ordenadores son de algún profesor (de manera que es más normal que un profesor no tenga ordenador, mientras que es menos normal que un ordenador no tenga un profesor asignado), podemos pensar que es un caso que está cerca de:

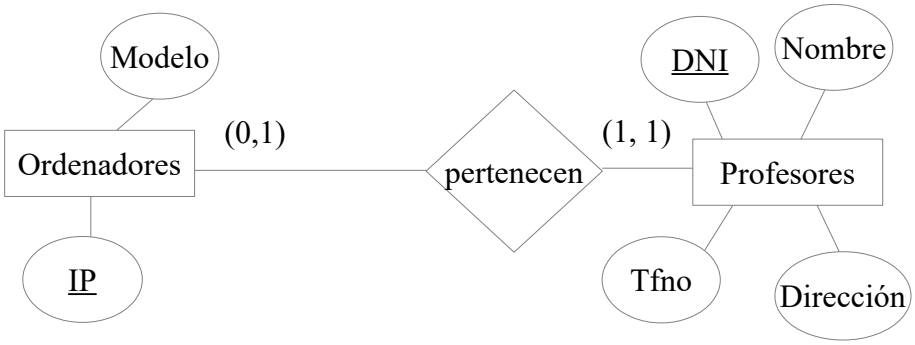


Figura 26: Imaginando un caso (0,1) - (0,1) cercano al (0, 1) - (1, 1).

Y por tanto optaríamos por la solución de ese caso:

- Ordenadores añade un campo DNI que es clave ajena y unique (aunque no es NOT NULL, porque aunque improbable, sigue siendo posible tener ordenadores sin profesor).
- Este campo sólo vale NULL en los pocos casos en los que el ordenador no tenga profesor asignado, por lo que casi nunca es nulo. Esto es lo que hace a la solución ventajosa, pues hemos minimizado el número de posibles nulos.

4.4.3 En los dos lados la cardinalidad es (0,1) y es excepcional que los elementos participen en la interrelación (+)

Supongamos el ejemplo de la Figura 27, en el que modelamos que los empleados de un banco, por cuestiones de movilidad, a veces viven en alguno de los inmuebles que son propiedad del banco quizás consecuencia de un desahucio.

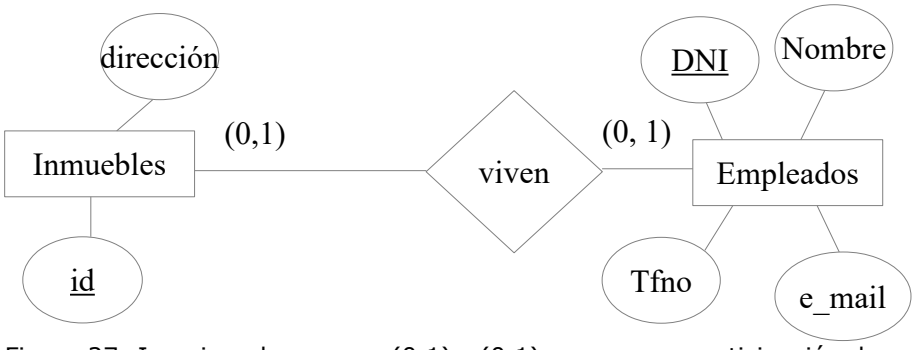


Figura 27: Imaginando un caso (0,1) - (0,1) con escasa participación de las entidades en la interrelación.

Como hemos visto en la sección anterior, podemos implementar la interrelación mediante una clave ajena con el *id* del inmueble en la tabla *Empleados*, pero como casi ningún empleados vive en un inmueble del banco ese campo estaría casi siempre valiendo *null*. Pero si implementamos llevando el DNI del empleado como clave ajena a la tabla de *Inmuebles* nos pasa lo mismo, pues son raros los inmuebles dedicados a albergar a empleados, por lo que ese campo DNI casi siempre valdría nulo.

En [Elmasri y Navathe 2007] sección 7.1.1. Paso 3 proponen crear una tabla aparte para representar las ocurrencias de empleados que viven en los inmuebles del banco, en lo que llaman la *"metodología de referencia cruzada o relación de ración"*. Esto es:

```
CREATE TABLE inmuebles(
     id        NUMERIC(6) PRIMARY KEY,
     direccion      VARCHAR(80));

CREATE TABLE empleados(
     DNI  NUMERIC(8) PRIMARY KEY,
     nombre    CHAR(50) NOT NULL,
     tfno NUMERIC(9),
     email     CHAR(80));

CREATE TABLE viven(

     IdInmueble NUMERIC(6) PRIMARY KEY
                              REFERENCES inmuebles,
     DNIempleado NUMERIC(8) UNIQUE NOT NULL
                              REFERENCES empleados
);
```

En la tabla *viven* las dos claves ajenas son claves candidatas, evitando que se repita el DNI (i.e., que un empleado tuviera más de un inmueble), y que se repita el *id* del inmueble (i.e., evitando que en el mismo inmueble vivieran varios empleados).

Con esta solución se ha evitado la aparición de valores nulos a consecuencia de la implementación de la relación.

4.5. Cardinalidades Muchos a Muchos (*)

Hasta ahora nos hemos centrado en las interrelaciones 1:N, y en las 1:1 como caso especial de las 1:N. El modelo E/R permite representar también interrelaciones muchos a muchos (M:N). Tomemos el siguiente ejemplo:

Figura 28: Ejemplo de Interrelación N:M en el E/R.

Un autor puede escribir varios libros, y un libro puede ser escrito por varios autores. Por tanto, a cada libro le pueden corresponde como máximo *n* autores; lo que causa que la cardinalidad máxima de la izquierda sea *n*. De la misma forma, a cada autor le corresponden como máximo *n* libros; lo que se representa mediante la *n* de la derecha.
Por tanto, en una interrelación M:N las cardinalidades máximas son siempre *n*, pero las mínimas pueden tomar tanto el valor 0, como el 1. En nuestro ejemplo se dan los dos casos. Hemos supuesto que un autor tiene que escribir al menos un libro obligatoriamente: (1, n). Sin embargo, un libro parece que podría no tener autores, quizás porque fuese anónimo, por ello figura la cardinalidad (0, n).

4.5.1 Atributos en las Interrelaciones M:N (*)

Si buscamos alguna definición de atributo en el modelo E/R, podemos encontrarnos cosas como esta: "*propiedad de un conjunto de entidades o un* ***conjunto de interrelaciones*** *caracterizada por un nombre y un tipo elemental*" [Gardarin 1993].
Esto significa, que los atributos pueden asignarse a las interrelaciones. Las interrelaciones 1:N excepcionalmente podrían en algún caso llevar atributos. Normalmente las interrelaciones 1:N no llevan atributos, porque en todo caso es posible siempre argumentar que el atributo no es de la interrelación, sino de la entidad del extremo *a varios*[7].
Por ejemplo, supongamos que un profesor-tutor puede tutorar varios alumnos, y un alumno tiene un solo profesor-tutor como mucho, y que además, la mayoría de los alumnos no tienen tutor. Nos interesa registrar en la base de datos la hora y día de la semana de la tutoría. Cada alumno es atendido por separado por el profesor en un momento distinto de la semana. El diagrama E/R resultante sería:

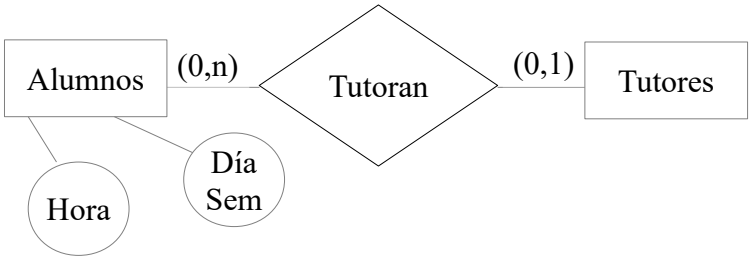


Figura 29: Atributos en las Interrelaciones 1:N

Es lógico pensar que la hora y día de la semana es un atributo de los alumnos. Si fuera del profesor, significaría que todos sus alumnos son atendidos simultáneamente. Como cada alumno tiene un único día y hora, podemos hacer que sean atributos de alumnos.

En definitiva, en relaciones 1:N y 1:1 normalmente no tendremos atributos.[8]

Sin embargo, en las interrelaciones N:M la aparición de atributos propios es habitual, no tiene nada de excepcional; y cuando un atributo es de la interrelación no hay duda alguna de que no puede ser de alguna de las dos entidades que conecta.

7 No obstante, si el diseñador estima que de esa forma se representaba mejor la semántica del problema podría asignar atributos a una interrelación 1:N, aunque insistimos en que es una situación raramente justificable.

8 En [de Miguel y Piattini 1993] sección 9.1.1 aparece un contraejemplo a esta regla para la interrelación 1:1 *matrimonio* entre las entidades *marido* y *mujer*, de manera que el atributo *fecha del matrimonio*, no parece oportuno ponerle en ninguna de las 2 entidades, sino en la interrelación; pero no deja de ser un caso extremadamente excepcional.

Por ejemplo, supongamos dos conjuntos de entidades: *Alumnos* y *Asignaturas*, interrelacionados por la interrelación *estar-matriculado-de* que representa de qué asignaturas está matriculado un alumno actualmente. Como un alumno puede matricularse de varias asignaturas, y una asignatura típicamente tendrá varios alumnos, es una interrelación N:M.

Las cardinalidades mínimas suponemos que son cero en ambos casos porque a) podemos suponer que hay asignaturas optativas de las que no se matricule ningún alumno, y b) puede haber situaciones puntuales en el sistema de información en las que el alumno no esté matriculado de ninguna asignatura, como por ejemplo la situación de un alumno de segundo entre la fecha de cierre de actas y antes del inicio de la matrícula.

Supongamos que queremos almacenar en la base de datos la nota que el alumno obtiene en la asignatura mediante un atributo *nota*. Si la nota fuese un atributo de la asignatura, todos sus alumnos tendrían la misma nota. Si la nota fuese un atributo del alumno, querría decir que cada alumno siempre obtiene la misma nota en todas sus asignaturas. Así pues, no puede tratarse de un atributo de ninguna de las entidades.

Recordemos que la interrelación es un conjunto cuyos elementos son "*las flechas*" que unen cada elemento de una de las entidades, con un elemento de la otra. Por ejemplo, si *Luis* está matriculado de *Ofimática*, habrá un elemento (i.e, una flecha) en la interrelación *estar-matriculado* que represente ese par alumno-asignatura. Observemos que la nota no es del alumno, ni de la asignatura, sino de ese par alumno-asignatura, ya que cada alumno puede sacar una nota distinta en cada asignatura, y en cada asignatura las notas de cada alumno pueden ser distintas. Por tanto, no hay duda de que este atributo tiene que ser de la interrelación, lo cual representaremos como en la siguiente ilustración.

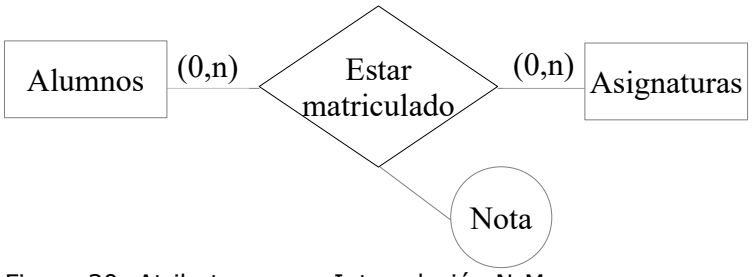


Figura 30: Atributo en una Interrelación N:M.

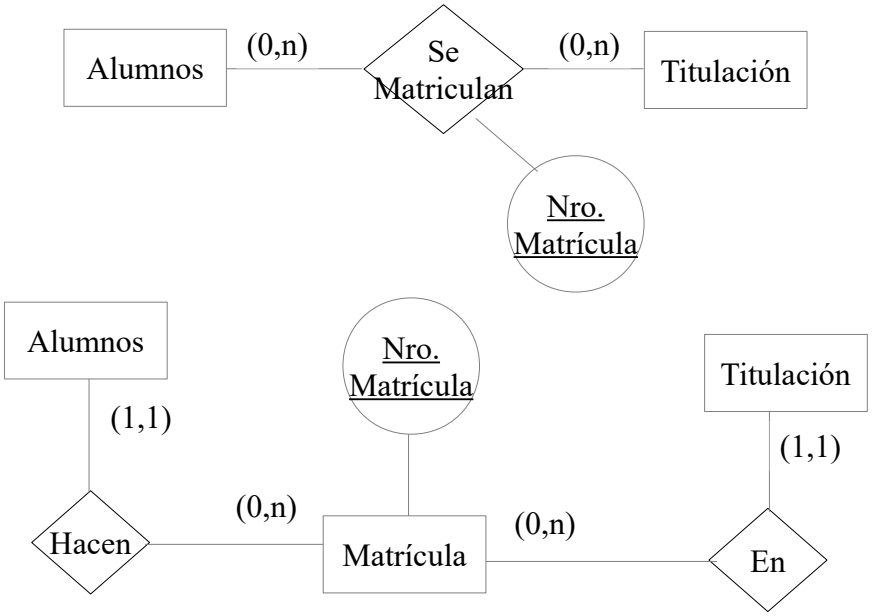


Figura 31: Las interrelaciones nunca llevan AIP. **La figura de arriba es errónea**. Si sospechamos que la interrelación lleva a AIP habrá que transformarla en una entidad, como en la parte de abajo.

Finalmente, resaltaremos que **una interrelación, aún teniendo atributos, no tiene AIPs ni AIAs**. Si los tuviera significaría que cada elemento de la interrelación tendría sentido por si mismo, puesto que sería capaz de identificarse sin contar con las entidades que conecta. Esto, a la postre, significa que la presunta interrelación es seguramente una entidad, las cuales si pueden tener AIPs y AIAs, y que por tanto, estamos modelando el problema erróneamente. Como vemos en el ejemplo, el atributo número de matrícula, indica que la matrícula se identifica por si misma, sin necesidad del alumno o la titulación, por lo que debe considerarse una entidad, tal y como hemos representado abajo. Ahora bien, no tiene sentido una matrícula no relacionada con un alumno o con una titulación, por lo que las cardinalidades mínimas que lo relacionan con estas entidades son uno. Es decir, la matrícula depende por existencia de alumnos y titulación.

4.5.2 Discusión: ¿Cuando es atributo?, ¿cuándo es entidad? (*)

A veces tenemos diagramas E/R en los que no sabemos si un determinado elemento es un atributo o una entidad. Con todo lo visto podemos tener un mayor criterio para responder a esta pregunta. Tenemos varios indicadores al respecto:

1. Si el elemento tiene existencia propia, es una entidad siempre. En algunas ocasiones incluso podría ser una entidad a pesar de no tener existencia propia (entidad subordinada, o una débil en una relación de dependencia).
2. Si ese elemento candidato a ser atributo, tiene a la vez sus propios atributos, entonces: o es un atributo compuesto o es una entidad. En general casi nunca utilizaremos atributos compuestos, por lo que es un buen síntoma de que estamos ante una entidad.
3. Si ese elemento candidato a ser atributo, toma varios valores al relacionarlo con otro elemento del diagrama, entonces o es multivaluado o es una entidad que a su vez es la parte “a uno” de una relación 1:N, o es un extremo de una N:M . En general, si tiene un sólo atributo podemos pensar en la posibilidad del atributo multivaluado, pero si tiene varios, lo normal es considerarlo una entidad, ya que en la práctica nadie usa atributos multivaluados compuestos (recarga en exceso el diagrama).
4. Si ese elemento candidato a ser atributo, participa por su cuenta en otras interrelaciones ha de ser una

entidad. Por ejemplo, si un alumno se matricula de un curso y sólo sabemos el nombre del curso, curso será atributo, pero si curso está relacionado con los profesores que lo imparten es entidad.

Un caso típico de discusión es el de las provincias. Por ejemplo, los alumnos son de una determinada provincia de la que sólo guardamos el nombre de dicha provincia. Como se trata de un solo atributo parece que es mejor tratarlo como atributo. Como no nos importa que cuando no haya alumnos de una provincia, la provincia no figure en la base de datos, las provincias no tendrían existencia propia. Nuevamente otro indicio de que es un atributo. Los alumnos son de una única provincia, por lo que en todo caso no es multivaluado.

Lógicamente, si las provincias estuviesen relacionadas con algún elemento más de la base de datos, (p.e. Los centros, los profesores ...) deberíamos considerarla entidad. Supongamos que no es así: las provincias no se relacionan con nadie mas. Aún así podría interesarnos hacerla entidad, pues puede que sea interesante dotarle de existencia propia (i.e., que la provincia aparezca en la base de datos aunque aún no tenga alumnos), para que desde, por ejemplo, un formulario de matriculación, el alumno pueda elegir su provincia desde un desplegable.

4.5.3 Observación general sobre la cardinalidad mínima 1

La asignación de las cardinalidades mínimas no es trivial y a veces ha de tener en cuenta ciertos matices que normalmente tienen que ver con **estados iniciales**. Por ejemplo, estamos de acuerdo en que un departamento debe de tener al menos un profesor, pero ¿eso va a ser así siempre en la base de datos que soporte al sistema de información que estamos diseñando?, o por el contrario ¿se va a permitir que los usuarios den primero de alta los departamentos antes de asignarlos a algún profesor?.

En esta decisión de cómo se gestiona el "estado inicial" correspondiente a la creación o alta de un departamento radica si finalmente ponemos un 0 o un 1 en la cardinalidad mínima de la izquierda de la figura. Y, aunque no tenga sentido tener departamentos sin profesores, si en el momento de darlo de alta en la base datos no hace falta, o no hay que asignarlo a ningún profesor, deberemos de colocar un cero en la cardinalidad mínima.

En general las cardinalidades mínimas casi siempre serán cero, y cuando pensemos que es uno, deberemos de repasar ciertas circunstancias especiales que normalmente tienen que ver con el momento en el que se crea la entidad. Por ejemplo, en la Figura 28, la interrelación N:M entre *autores* y *libros* es (1,n) en el extremo de los *libros,* pero aunque pensemos que no es lógico ser autor sin antes haber escrito un libro, a efectos de nuestra base de datos, el cliente nos ha podido especificar su deseo de que sea posible introducir los datos de una autor sin tenerle que asignar todavía ningún libro, permitiendo que el usuario en otro momento decida qué libros asignar a ese autor.

Figura 32: Ejemplo de Interrelación N:M en el E/R. Teniendo en cuenta estados iniciales.

En este caso es más adecuado el diseño de la Figura 32. Por el contrario, si lo que quiere el cliente del sistema de información es que tanto en el alta de un autor, como en cualquier otro momento, no se permita que un autor no tenga asignado ningún libro, lo más adecuado sería un diseño como el de la Figura 28, pero esa restricción puede correr en contra del usuario y a la larga convertirse en un inconveniente.

Otros ejemplos de lo mismo: ¿puedo dar de alta la línea telefónica antes de asignarla ningún profesor?, ¿puedo dar de alta al recluso antes de asignarle una celda?, ¿puedo dar de alta el departamento sin todavía nombrar al director?. La respuesta a estas preguntas las tiene nuestro cliente, pero nosotros le deberemos de abrir los ojos de las consecuencias de las mismas en cuanto al funcionamiento de los programas que trabajen contra la base de datos: ¡cuidado!, luego no se queje de que no puede dar de alta la línea telefónica hasta que no estén dados de alta sus profesores; de que no

puede dar de alta el recluso hasta que no esté claro qué celda le corresponde, de que no puedo dar de alta el departamento hasta que no estén los resultados de las elecciones a director etc...

4.5.3.1 Cardinalidad mínima 1 en los dos extremos

Una derivada de esta discusión es que aunque teóricamente podríamos encontrarnos interrelaciones binarias con cardinalidad mínima "1" en ambos extremos, en la práctica, y teniendo en cuenta los estados iniciales, no es una situación muy realista, porque alguno de los dos extremos de la interrelación habrá de ser insertado primero en la base de datos.

El caso 1:1 es especial, se discutió en las secciones 4.3. y 4.4.1, y se vió en definitiva de que probablemente se tratase de una sóla entidad, y que en cualquier caso, derivaría en una sola tabla que aglutinaba los campos de las dos entidades conectadas por esa interrelación.

Los casos 1:N y N:M, sin embargo si que obligan a que en un estado inicial uno de los dos extremos se inserte primero sin estar relacionado con un elemento del otro extremo. Por ejemplo:

1. En la Figura 17, el caso de una interrelación 1:N entre *departamentos* y *profesores*, parece que hay que insertar alguno de los dos elementos primero. En este caso parece claro que son los *departamentos,* y que más bien el lado de los *profesores* sería entonces (0, n) y no (1, n).

- En la Figura 33, N:M entre *profesores* y *asignaturas* si que parece un sinsentido que haya profesores sin asignaturas o asignaturas sin profesores. Pero nuevamente, alguno de los dos elementos tiene que insertarse primero en la base de datos, por ejemplo las asignaturas, y entonces durante un tiempo esa asignatura estará impartida por cero profesores.

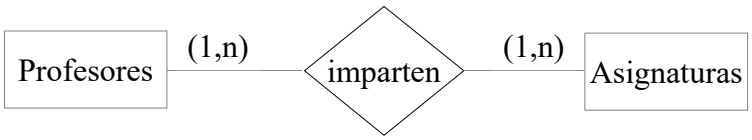


Figura 33: Ejemplo de Interrelación N:M en el E/R con cardinalidad mínima 1 en ambos extremos.

Por tanto, en la práctica seremos precavidos y escépticos con con esos casos que tienen cardinalidad mínima uno en ambos lados.

En cualquier caso, aunque hubiese cardinalidad mínima 1 en ambos lados, no podemos controlarla únicamente con las restricciones del CREATE TABLE (habría que utilizar *triggers* o ASSERTIONs[9] que ya veremos por encima más adelante en la sección 6.2.2). Por tanto, lo único que podemos hacer:

1. En el caso 1:N actuar como si el (1, n) del lado "a varios" fuese un (0,n).
2. En el caso N:M actuar como si las dos cardinalidades (1, n) fuesen (0,n).

4.6. Paso al Modelo Relacional de las Relaciones M:N

Para pasar una interrelación M:N a tablas:

1. Cada una de las entidades que conecta la interrelación pasa a ser tabla, y no se lleva a ninguna de estas tablas, ningún atributo extra en forma de clave ajena, como hacíamos en el caso 1:N.
2. Toda interrelación varios a varios se convierte en tabla, de forma que:
    1. Los atributos de la interrelación pasan a ser atributos de la tabla.
    2. Los atributos identificadores principales de las entidades que conecta, pasan a ser atributos de la tabla.
    3. Los atributos identificadores principales de las entidades que conectan, pasan a formar en conjunto, la clave primaria compuesta de la tabla resultante de la interrelación.

9 En el Glosario parece una explicación detalla de lo que es uno objeto ASSERTION, con el correspondiente ejemplo.

4. Los atributos identificadores principales de la entidades que conectan, pasan a formar por separado, las claves ajenas que relacionan esa tabla con cada una de las tablas de las entidades que conecta.

Desde el punto de vista de la declaración SQL, como la existencia de filas en la tabla de la interrelación no tiene sentido, sino es para expresar la relación entre dos instancias de las entidades:

1. Cada uno de los AIPs de las tablas que conecta, en la interrelación han de ser NOT NULL, pero no hará falta ponerlo, ya que forman parte de la clave primaria.
2. Es conveniente (no imprescindible) declarar estas claves ajenas con ON DELETE CASCADE y ON UPDATE CASCADE, ya que las filas de la tabla correspondiente a la interrelación dependen de que existan las instancias de entidad que conectan.

Supongamos el ejemplo de la Figura 30, la declaración de tablas que le corresponde sería:

```
CREATE TABLE estarMatriculado (
 IdAlumno  CHAR(10) REFERENCES ALUMNOS
           ON DELETE CASCADE ON UPDATE CASCADE,
 IdAsignatura   CHAR(10) REFERENCES ASIGNATURAS
           ON DELETE CASCADE ON UPDATE CASCADE,
 Nota      NUMERIC(3,1),
 PRIMARY KEY ( IdAlumno, IdAsignatura )
);
```

Supón ahora que las asignaturas en lugar de identificarse por *IdAsignatura*, se identifican por el AIP compuesto que surge de la combinación de *IdTitulación* con *IdAsignatura.* Entonces el resultado sería:

```
CREATE TABLE estarMatriculado (
 IdAlumno  CHAR(10) REFERENCES ALUMNOS
           ON DELETE CASCADE ON UPDATE CASCADE,
 IdTitulación   CHAR(3),
 IdAsignatura   CHAR(5),
 Nota           NUMERIC(3,1),
 PRIMARY KEY ( IdAlumno, IdTitulación, IdAsignatura ),
 FOREIGN KEY ( IdTitulación, IdAsignatura )
           REFERENCES Asignaturas
           ON DELETE CASCADE ON UPDATE CASCADE
);
```

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

***Ejercicio**: Haz un diagrama E/R genérico correspondiente a dos entidades interrelacionadas por una relación varios a varios. Utiliza letras para designar los nombres de las 2 entidades, la relación y los atributos que necesites. Deduce las dependencias funcionales y las claves. ¿A que forma normal llegas y por qué?.*

5. Dependencias en el Diagrama E/R (*)

5.1. Dependencia por Existencia en el E/R (*)

La dependencia por existencia es un fenómeno que ocurre cuando hay una relación uno a muchos (excepcionalmente también podría ser uno a uno, al ser el 1:1 un caso especial del 1:N) entre dos conjuntos de entidades, de manera que las entidades de la parte a muchos no pueden existir por si solas, necesitan que exista una instancia de la entidad de la parte "a uno" con la que relacionarse.

Se dice que la entidad de la parte "a uno" es ***dominante***, y la otra (i.e., la dependiente, la de la parte "a varios") es ***subordinada***.

En [Connolly y Begg 2005] sección 11.6.5 esta dependencia viene explicada de manera equivalente como *restricción de participación,* en cuanto "*determina si todas las instancias de entidad participan en una relación o sólo lo hacen algunas*".

Por ejemplo, supongamos una base de datos con dos entidades: *profesores* y *cuentas de correo*. Cada profesor puede tener varias cuentas de correo, quizás ninguna; pero cada cuenta de correo corresponde a un profesor y solamente uno. Esto hasta ahora lo habíamos representado de la siguiente forma:

Figura 34: Dependencia por existencia denotada por la cardinalidad mínima en la parte a 1.

Y ya en esta representación el "1" de la cardinalidad mínima de la izquierda del (1,1) está indicando claramente que un e-mail no puede existir sin estar relacionado con al menos un profesor.

Algunos autores, innecesariamente, añaden algún tipo de marca para reforzar visualmente la existencia de esa dependencia. Por ejemplo, Piattini [de Miguel y Piattini 1993] añade una "*E*" (de Existencia) en la parte superior del rombo y la flecha apuntando al lado "a varios", tal que así:

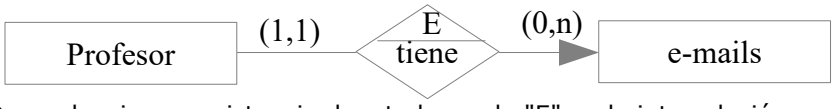


Figura 35: Dependencia por existencia denotada por la "E" en la interrelación.

Otros [Chen 1976] sólo hubieran representado la flecha, otros haciendo que la línea entre el rombo y *e-mails* fuese doble y sin flecha [Date 2000], [Elmasri y Navathe 2007], [Silberschatz et. al 2006], etc … Nosotros podemos también adoptar cualquiera es estos estilos, pero a sabiendas de que no ganamos expresividad en el diagrama_ **mientras usemos cardinalidades mínimas**.

5.1.1 Pasar a tablas una dependencia por existencia

Pasar a tablas una dependencia por existencia a tablas SQL no es muy diferente que pasar una relación uno a varios. Es una buena ocasión para utilizar ON DELETE/UPDATE CASCADE, si bien no es estrictamente necesario, ya que con NO ACTION también preservamos que una fila de la parte a varios no se quede huérfana.

La diferencia entre que haya dependencia por existencia, y no la haya estaría en que en la tabla hija no debería de poder haber filas no relacionadas con la tabla padre, y eso se controla:

1. Declarando la correspondiente FOREIGN KEY (quizás con CASCADE), hasta aquí no habría diferencia con la no-dependencia por existencia, y
2. Declarando que todos los campos de esa clave ajena son NOT NULL, y esta sí es la diferencia.

Así, para pasar a tablas el ejemplo de la Figura 35, en el que las cuentas de correo dependían por existencia de los

profesores, supuesto que los profesores se identificasen por una clave compuesta *nProfesor* + *nDepartamento*, haríamos lo siguiente:

```
CREATE TABLE profesor (
     nProfesor      INTEGER,
     nDepartamento  INTEGER,
     nombre         CHAR(30),
     PRIMARY KEY ( nProfesor, nDepartamento)
);

CREATE TABLE eMails (
     cuenta         CHAR(50) PRIMARY KEY,
     fechaAlta DATE DEFAULT CURRENT_DATE NOT NULL,
     fechaBaja DATE,
     nProfesor INTEGER        NOT NULL,
     nDepartamento  INTEGER   NOT NULL,
     FOREIGN KEY ( nProfesor, nDepartamento )
                         REFERENCES profesor
);
```

Donde vemos que en la última tabla todos los campos de la clave ajena con *profesores* son no nulos.
Opcionalmente, podríamos haber añadido opciones en cascada a la tabla *eMails*, pero siendo conscientes de que estrictamente el borrado/modificación en cascada no controlan la dependencia por existencia, si bien ayudan a darla matices adicionales, que por otro lado se podrían dar también a una relación sin dependencia por existencia.
Por ejemplo, podríamos haber añadido una restricción de borrado en cascada:

```
CREATE TABLE eMails (
     cuenta         CHAR(50) PRIMARY KEY,
     fechaAlta DATE DEFAULT CURRENT_DATE NOT NULL,
     fechaBaja DATE,
     nProfesor      INTEGER NOT NULL,
     nDepartamento  INTEGER NOT NULL,
     FOREIGN KEY ( nProfesor, nDepartamento )
          REFERENCES profesor ON DELETE CASCADE
);
```

5.2. Dependencia por Identificación en el Diagrama E/R (*)

Hasta ahora habíamos visto que todo conjunto de entidades tenía un AIP (atributo identificador principal). Pues bien, existen casos excepcionales en los que no es así. En dichos casos, el conjunto de entidades sin AIP estará relacionado al menos con un conjunto de entidades que servirá para identificar en alguna medida sus instancias.

El caso típico es una relación binaria uno a muchos, en el que en la parte "a muchos" estará este conjunto de entidades sin AIP, al que llamaremos ***entidad débil***. Al otro extremo de la interrelación (en el que sí habrá AIP, la parte "a uno") lo llamaremos ***entidad fuerte***[10].

Por ejemplo, las facturas tienen un AIP, *numFactu*, y varios atributos, p.e: *fecha* e *importe total*. Cada factura se relaciona con varias líneas de facturas o línea de detalle. Cada línea de factura tiene un concepto, una cantidad y un importe. El diagrama E/R resultante será:

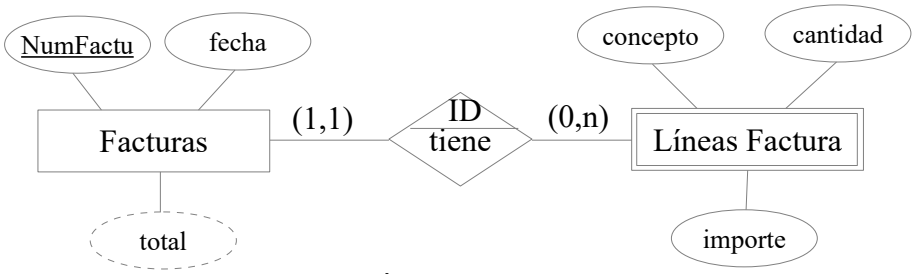


Figura 36: Dependencia por Identificación denotada por el ID en el rombo. La entidad débil se marca con un recuadro doble.

Vemos que:

1. La entidad débil se representa con un doble recuadro. Algunos autores también ponen doble línea en al arco que une a las entidades subordinadas (e.g., [Date 2000], [Elmasri y Navathe 2007], [Silberschatz et. al 2006]), (ver siguiente ilustración).

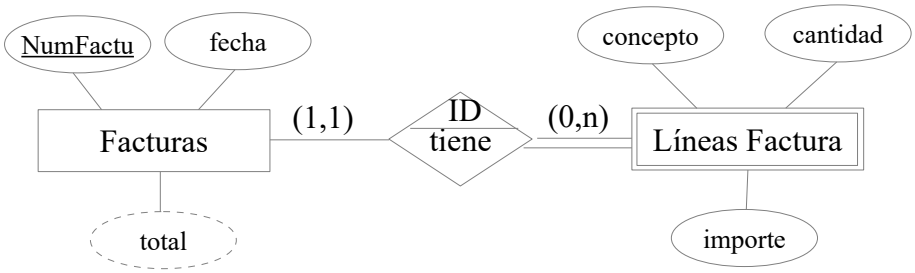


Figura 37: Doble arco que utilizan algunos autores para remarcar a la parte débil de una dependencia por identificación.

2. La interrelación algunos autores la etiquetan con ID [de Miguel y Piattini 1993]. Algunos autores, como el propio Peter Chen, [Chen 1976], inventor del diagrama E/R, en lugar de marcar con un ID la interrelación, la marcan con un doble rombo (también [Date 2000], [Elmasri y Navathe 2007], [Ullman y Widom 1997]). MÉTRICA [METRICA V3 2001] no aclara nada al respecto, pero sin embargo, es imprescindible algún tipo de notación, pues si la entidad débil fuese extremo "a varios" de 2 o más interrelaciones 1:N, sin esa notación no estaría claro de cuál depende. Nosotros vamos a mantener la notación con el ID de [de Miguel y Piattini 1993].
3. La entidad fuerte es siempre el lado "a uno", y la débil el lado "a varios". La cardinalidad en el lado a uno es siempre 1,1. De manera que podemos pensar que la dependencia por identificación es un caso especial de dependencia por existencia. O dicho de otra forma: **siempre que hay dependencia por identificación,**

10 Normalmente una entidad débil depende de una única entidad fuerte. Sin embargo, existen en la bibliografía autores que contemplan la posibilidad de que la misma entidad débil tenga varias fuertes (e.g., [Elmasri y Navathe 2007] pg 76, [Ullman y Widom 1997] pgs 73-74)

la hay por existencia. Lo contrario no es cierto.

El atributo *total* está modelado como derivado, pues se supone que se obtiene del sumatorio de los productos de *importe* * *cantidad*, por ejemplo.

Supongamos, una línea de factura por la compra de 5 lapiceros a 30 céntimos cada uno en la factura 100. Si en la factura 200 también hay una línea de factura similar, probablemente seas de los que piensan que se trata de la misma línea, y no es así, pues si el cliente de la factura 100 decide comprar un lápiz más, y ambas líneas de factura fuesen la misma al actualizar la linea de factura, el cliente de la factura 200 tendría que comprar un lápiz que no quiere. Por eso la línea de factura "5 lapiceros a 30 céntimos cada uno" de la factura 100 no puede ser en la base de datos la misma que en la factura 200, y lo que las diferencia, es precisamente la factura a la que pertenecen.

Las entidades débiles tendrán un conjunto de atributos (normalmente formado por un solo atributo) llamado ***discriminante*** que subrayaremos con línea discontinua. El discriminante unido al AIP de la entidad fuerte tendrá un valor único (irrepetible) para cada elemento de la entidad débil.

En el ejemplo anterior, supongamos que el enunciado nos dice que no puede haber dos *conceptos* iguales en la misma factura. Entonces, el atributo *concepto* sería el discriminante, con lo que finalmente el diagrama E/R quedaría tal que así:

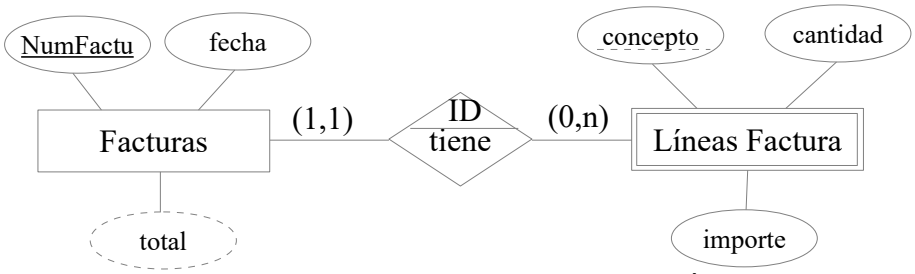


Figura 38: El subrayado discontinuo representa que concepto actúa de discriminante en la entidad débil Líneas de Factura.

Podemos encontrar algún otro caso más o menos curioso. como el siguiente ejemplo de dependencia en cascada:

> En una red de ordenadores, los *dominios* tienen un *nombre de dominio* único, las *máquinas* pertenecen a un dominio, y se identifican a través de su *nombre* y el *nombre de dominio*. Puede haber dos nombres de máquinas iguales, pero estarían en dominios distintos. Las *cuentas de usuario* de cada máquina son distintas, puede haber dos cuentas iguales, pero en máquinas distintas.

Quedaría algo así (el atributo *IP* de máquinas se refiere al último campo de la dirección IP, mientras que *rangoIPs* representa los tres primeros):

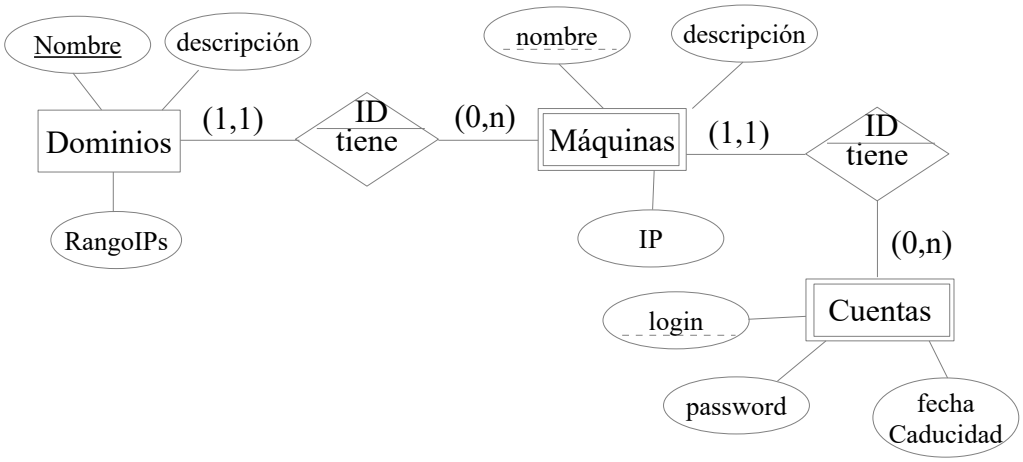


Figura 39: Ejemplo de dependencia por identificación en cascada.

Vemos que cuentas depende de máquinas y ésta de dominios. Lo interesante es ver que:

1. No puede haber dos máquinas que tengan el mismo *nombre de dominio* y a la vez el mismo *nombre de máquina*.
2. No puede haber dos cuentas de usuario con el mismo *login* en la misma *maquina*, y por tanto no se puede repetir en la base de datos la tripleta (*nombre de dominio, máquina, login*).

La IP de la máquina podía haber sido elegida alternativamente como discriminante, por ello forma una clave candidata no primaria al juntarla con el dominio en la tabla de máquinas. El E/R no tiene una notación para discriminantes alternativos, por ello no hemos podido marcarlo en el diagrama. Es en el fondo el mismo problema que teníamos para denotar los AIAs.

5.2.1 Pasar a tablas una dependencia por identificación

El mayor problema que surge al pasar una dependencia por identificación a tablas relaciones es identificar la clave primaria de la entidad débil. Como ya sabemos, para ser consistentes con el modelo relacional, toda tabla debería de tener clave primaria.

En el ejemplo de la Figura 38, en el que hemos obviado el campo derivado *Total* en la tabla de *facturas*, se observa que podría haber dos líneas de factura con el mismo *concepto*, *cantidad* e *importe.* Lo normal es que esas líneas correspondiesen a facturas diferentes, de manera que parece que no hay un conjunto de atributos de *Línea de Factura* que pudiesen formar por si mismos la clave primaria.

En el ejemplo anterior, el discriminante *concepto* nos dice que no puede haber dos conceptos iguales en la misma factura. Por tanto, el discriminante unido a la clave ajena de la relación si que forma una clave primaria de la tabla generada por la entidad débil. En nuestro caso, el concepto unido a *NumFactu* identifica unívocamente a cada línea de factura. Esto daría lugar a la siguiente declaración de tablas SQL:

```
CREATE TABLE facturas (
     NumFactu  INTEGER PRIMARY KEY CHECK(numFactu >=0),
     fecha     DATE
);

CREATE TABLE LineasFactura (
     NumFactu  INTEGER REFERENCES facturas
               ON DELETE CASCADE ON UPDATE CASCADE,
     concepto  CHAR(10),
     cantidad  INTEGER CHECK( cantidad > 0),
     importe   NUMERIC ( 10, 2),
     PRIMARY KEY (NumFactu, concepto)
);
```

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

Es importante observar:

1. Que la clave primaria de la tabla de la entidad débil es una clave compuesta por la clave de la fuerte más el discriminante. Como consecuencia, la clave ajena que relaciona a la entidad débil con la fuerte forma parte de la clave primaria. Todo esto será siempre así en cualquier entidad débil.

2. Que normalmente se declaran las acciones en cascada, si bien no es estrictamente necesario. Recuerda que estamos en un caso especial de dependencia por existencia, luego aplicamos el mismo criterio que entonces.
3. Que en la clave ajena no nos ha hecho falta poner el NOT NULL, como hacíamos en la dependencia por existencia. La clave ajena, como en la dependencia por existencia, sigue teniendo que ser NOT NULL, pero la diferencia ahora es que como pertenece a la clave primaria, y la restricción de clave primaria implica NOT NULL, ya no hace falta ponerlo.

Como ejemplo adicional vemos qué pasa cuando una entidad débil depende a su vez de otra débil. Recordemos el ejemplo de la Figura 39.

Vemos que *cuentas* depende de *máquinas* y ésta de *dominios.* Lo interesante es ver que la clave de la tabla de *cuentas* se va a construir a partir de su discriminante, que a su vez está compuesto por el discriminante de máquinas y la clave de dominios.

```
CREATE TABLE dominios(
     Nombre          CHAR(30)  PRIMARY KEY,
     descripción     CHAR(128),
     RangoIPs        CHAR(11) NOT NULL UNIQUE
);

CREATE TABLE maquinas(
     nombre          CHAR(15),
     dominio         CHAR(30) REFERENCES dominios
          ON DELETE CASCADE ON UPDATE CASCADE,
     descripción     CHAR(128),
     terminacionIP   NUMERIC(3) NOT NULL
               CHECK (terminacionIP>=0
                    AND
                    terminacionIP<=255),
     PRIMARY KEY ( nombre, dominio),
     UNIQUE ( terminacionIP, dominio)
);
```

```
CREATE TABLE cuentas(
     login          CHAR(15),
     dominio        CHAR(30),
     maquina        CHAR(15),
     password  CHAR(15) NOT NULL,
     fechaCaducidad DATE,
     PRIMARY KEY ( login, maquina, dominio),
     FOREIGN KEY ( maquina, dominio )
          REFERENCES maquinas
          ON DELETE CASCADE ON UPDATE CASCADE
);
```

Es importante en la clave ajena de *cuentas* poner *máquina* antes que *dominio*, porque en la tabla de máquinas la clave primaria se ha declarado poniendo primero el nombre de la *máquina* antes que el *dominio.*

Casualmente la terminación IP de la máquina podía haber sido elegida como discriminante, por ello forma una clave candidata no primaria al juntarla con el dominio en la tabla de máquinas.

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

***Ejercicio**: Haz un diagrama E/R genérico correspondiente a dos entidades interrelacionadas por una dependencia por identificación. Utiliza letras para designar los nombres de las 2 entidades, la relación y los atributos que necesites. Deduce las dependencias funcionales y las claves. ¿A que forma normal llegas y por qué?.*

5.3. Los Atributos Multivaluados (*)

Hasta ahora hemos considerado que los atributos de un conjunto de entidades tenían un único valor por cada instancia de entidad. Sin embargo el modelo E/R permite que haya atributos multivaluados, es decir, que tengan un conjunto o lista de valores.

Por ejemplo, supongamos que las personas tienen un conjunto de teléfonos, no sabemos cuantos, luego la solución de poner un número fijo *n* de atributos *teléfono1*, *teléfono2*, ... *teléfono_n*, no es válida. En ese caso podemos utilizar un atributo multivaluado *teléfonos*. Como vemos se denota por una elipse doble.

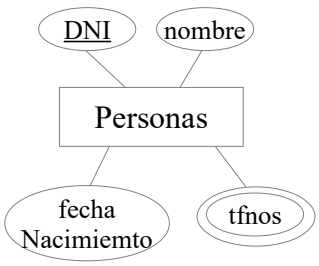


Figura 40: Notación de los atributos multivaluados por una elipse doble.

En realidad, hubiéramos tenido la misma expresividad haciendo el siguiente diagrama alternativo:

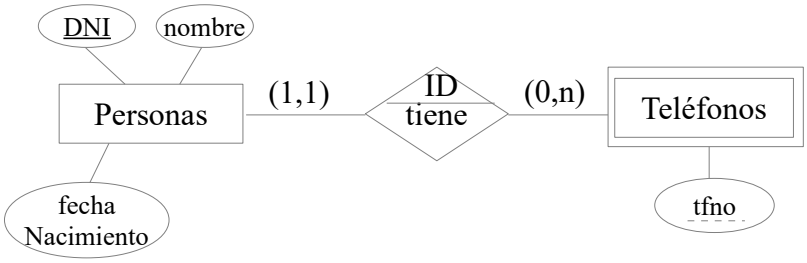


Figura 41: Diagrama E/R equivalente a otro con atributos multivaluados utilizando una entidad débil.

Donde vemos que el atributo que antes era multivaluado, actúa como discriminante, pues no es lógico que alguien tenga dos veces el mismo teléfono, aunque si puede haber varias personas que compartan el mismo.
Recuerda que el modelo relacional obliga a que toda relación esté normalizada, lo que significa que en el modelo relacional no hay atributos multivaluados.

5.3.1 Pasar a tablas los atributos multivaluados

Sabemos que los atributos multivaluados pueden ser representados mediante entidades débiles. Por ejemplo, el atributo multivaluado *teléfonos* de la Figura 40, puede verse en la Figura 41 como una entidad débil. En dicha representación vemos además que el atributo antes multivaluado actúa ahora como discriminante. Por lo tanto, para pasarlo a tablas, utilizaremos las reglas de paso correspondientes a las entidades débiles, obteniendo algo como esto:

```
CREATE TABLE personas (
     DNI            NUMERIC(8) PRIMARY KEY,
     nombre         CHAR(30) NOT NULL,
     fechNacimiento DATE
);

CREATE TABLE teléfonos (
     tfono     NUMERIC(9),
     DNI       NUMERIC(8) REFERENCES personas
               ON DELETE CASCADE ON UPDATE CASCADE,
     PRIMARY KEY (tfono, DNI)
);
```

En el caso de los multivaluados es de uso obligado la declaración de las restricciones CASCADE. No es lógico que al intentar borrar a una persona el sistema nos de un error porque tiene teléfonos, sino que lo adecuado es que le borre sin más junto con sus teléfonos.

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

***Ejercicio**: Haz un diagrama E/R genérico correspondiente a una entidad que tenga tanto atributos monvaluados como multivaluados. Utiliza letras para designar los nombres de la entidad y sus atributos. Deduce las dependencias funcionales y las claves. ¿A que forma normal llegas y por qué?.*

6. Roles e Interrelaciones reflexivas o recursivas (*)

Se dice que una interrelación binaria es ***reflexiva*** (o ***recursiva***) cuando ambos extremos de la interrelación son el mismo conjunto de entidades. Cuando ocurre ésto, es porque el conjunto de entidades está desempeñando dos papeles distintos en la misma relación, uno por cada extremo de la misma. Por ejemplo, supongamos una relación reflexiva entre empleados llamada "ser jefe de".

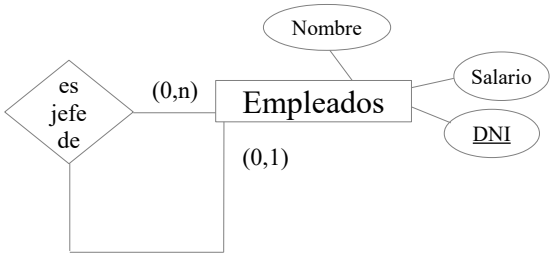


Figura 42: Ejemplo de interrelación reflexiva.

En un lado de la interrelación, los *empleados* están haciendo el papel de *jefes* (en el lado (0,1)), en el otro lado de *subordinados* (en el lado (0,n)). Estos papeles se conocen como roles. El ***rol*** es el papel que desempeña un conjunto de entidades en una interrelación.

Los roles se hacen notar con una etiqueta en el extremo correspondiente, por ejemplo:

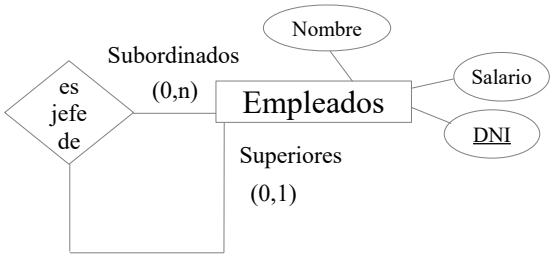


Figura 43: Ejemplo de interrelación reflexiva que adjunta las etiquetas correspondientes a los roles.

Las etiquetas de *superiores* y *subordinados* han sido colocadas en cada uno de los extremos de la interrelación, teniendo en cuenta que la parte "a uno" es la de los superiores, y la parte "a varios" la de los subordinados.

En realidad, podríamos colocar etiquetas de roles en cualquier interrelación, aunque no fuese reflexiva. Por ejemplo, en la interrelación *pertenece* entre profesores y departamentos los profesores desempeñan el el rol de pertenecer al departamento, y los departamentos, el rol de contener a los profesores, pero la información que aporta es muy obvia y en la práctica nadie lo hace. Sin embargo, en las relaciones reflexivas no es tan evidente qué papeles tiene una única entidad en una interrelación. El caso es más claro aún cuando la interrelación es aparentemente simétrica respecto a los dos extremos de la misma, como ocurre en las varios a varios. Por ejemplo, supongamos que queremos guardar en la base de datos de una compañía de autocares qué ciudades del país están interconectadas por sus líneas:

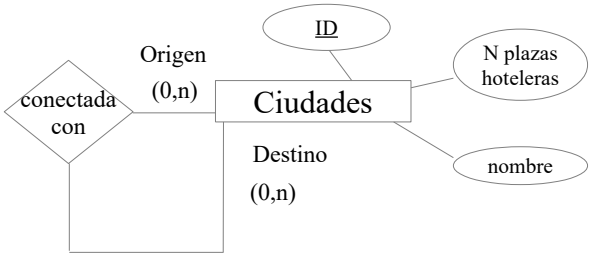


Figura 44: Interrelación reflexiva varios a varios.

Lógicamente, una ciudad puede ser orígenes de muchas conexiones, y destino de muchas conexiones. El que desde A se pueda ir directamente a B, no implica que desde B se pueda ir directamente a A, por lo que los roles en este caso si

son importantes, pues son distintos, y no hay nada en el diagrama que los permita deducirlos, salvo si los declaramos explícitamente.

6.1. Estructuras de datos representadas a través de interrelaciones reflexivas

¿Qué hay detrás de las interrelaciones reflexivas realmente?, ¿para qué sirven?. Las interrelaciones reflexivas sirven para representar ciertas estructuras de datos:

- Las interrelaciones reflexivas 1:1 sirven para representar listas enlazadas
- Las interrelaciones reflexivas 1:N sirven para representar árboles, y
- Las interrelaciones reflexivas N:M sirven para representar grafos dirigidos.

6.1.1 Listas enlazadas utilizando reflexivas 1:1

Una lista enlazada es un conjunto de nodos conectados por flechas, tal que a cada nodo le llega una flecha como máximo, y de cada nodo parte una flecha como máximo. Las figuras 45 y 46 representan dos tipos de listas enlazazas comunes. En la Figura 46 aparece una lista enlazada abierta, en la que hay un nodo principio que no recibe flechas y otro fin del que no sale ninguna, y en la Figura 47 hay una lista circular, en la que todo nodo recibe y envía una flecha siempre.

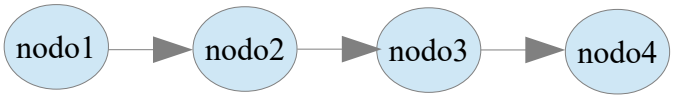


Figura 45: Lista enlazada abierta con 4 nodos.

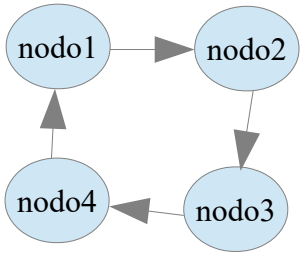


Figura 46: Lista enlazada circular con 4 nodos.

Estas dos estructuras pueden representar muchas situaciones de la realidad (e.g., una cola de un supermercado, un conjunto cíclico de tareas que se repiten etc …), y por eso se usan mucho en informática. En el modelo E/R, la forma de representar estas estructuras con las interrelaciones 1:1.

Supongamos que queremos representar sagas de películas (ejemplo tomado de [Ullman y Widom 1997]), de manera que cada película se relaciona de manera recursiva con la película que le sigue. Claramente, este ejemplo representa una lista.

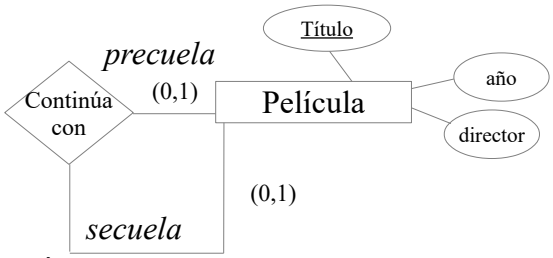


Figura 47: Interrelación reflexiva 1:1 representando una lista enlazada abierta.

A cada película:

1. La pueden seguir 0 películas como mínimo, porque por ejemplo no sea una película de una saga, o porque por ejemplo sea la última de la saga. Lo que nos lleva a una cardinalidad mínima 0 en el rol *precuela.*

2. La pueden seguir 1 película como máximo, pues sólo una película a los sumo puede ser la continuación de otra. Lo que nos lleva a una cardinalidad máxima 1 en el rol *precuela.*
3. El número de películas mínimo del que es continuación es 0, pues puede ser la primera plícula de la saga, o sencillamente que no se una película de una saga. Lo que nos lleva a una cardinalidad mínima 0 en el rol *secuela.*
4. El número de películas máximo del que es continuación es 1, pues a lo sumo, una película sólo puede ser continuación de otra. Lo que nos lleva a una cardinalidad máxima 1 en el rol *secuela.*

Observa que las cardinalidades mínimas son 0. Teóricamente podrían ser 1, pero siempre nos llevan a representaciones más artificiales de la misma información. Por ejemplo:

1. Si colocamos (1, 1) en el rol *precuela*, obligamos a que a toda película le siga otra. Esta situación nos puede parecer convincente si nuestra base de datos **sólo** almacena películas de saga, pero ¿a quién le sigue la última película de la saga?. ¿Cómo representamos a esa película que da fin a la saga?. Hay diversas soluciones:
   1. Que la última película de la saga sea continuada por si misma.
   2. Que la última película de la saga sea continuada por la primera. Esta última aproximación genera una lista circular, y nos obliga además a tener algún atributo extra que nos indique en qué parte de ese círculo empieza la saga.

   Cualquiera de las dos soluciones son menos elegantes que sencillamente permitir que una película, la última de la saga, no tenga una película siguiente.
2. Si colocamos (1, 1) en el rol *secuela*, obligamos a que a toda película le precedaotra. Esta situación también nos puede parecer convincente si nuestra base de datos **sólo** almacena películas de saga; pero ¿de quién es continuación la primera película de la saga?. ¿Cómo representamos a esa película que da comienzo a la saga?. Hay diversas soluciones, nuevamente:
   1. Que la primera película de la saga sea continuación de si misma.
   2. Que la primera película de la saga sea continuación de la última. Esta última aproximación genera nuevamente una lista circular, y nos obliga además a tener algún atributo extra que nos indique en qué parte de ese círculo empieza la saga.

   Nuevamente, parecen soluciones inapropiadas frente a dejar las cardinalidades mínimas a cero debido a que el concepto de fondo (i.e., las sagas cinematográficas) es una lista enlazada abierta.

Podrían existir situaciones en las que nos interese representar listas circulas. Pensemos en una base de datos de un sistema de información geográfica que guarda a través de los vértices de polígonos cerrados las manzanas de una ciudad (ver Figura 48).

Cada punto que es vértice de la manzana contiene las coordenadas x e y que a la vez son el propio AIP compuesto de la entidad *vértices de manzana*. A cada punto que es vértice de una manzana siempre le sigue uno y sólo uno, y siempre le precede uno y sólo uno, y no hay puntos iniciales ni finales, como ocurría en el ejemplo de las sagas cinematográficas, luego la representación como lista circular cerrada es la adecuada en este caso.

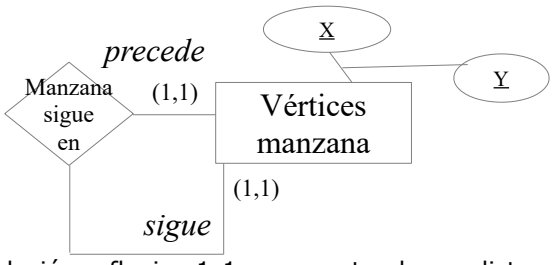


Figura 48: Interrelación reflexiva 1:1 representando una lista enlazada circular.

Por último, analicemos el caso de los matrimonios entre personas de la Figura 49. Las cardinalidades mínimas se han puesto a cero para dar cabida a los solteros, viudos y divorciados, que no están casados con nadie. La cardinalidad máxima es uno por no permitirse la poligamia.

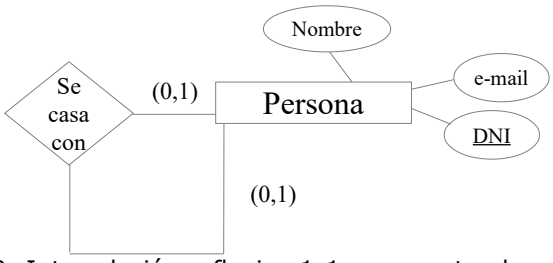


Figura 49: Interrelación reflexiva 1:1 representando parejas.

Si la entidad *Persona* la cambiamos por una entidad *Personas-Casadas*, porque la base de datos esté recogiendo únicamente un censo de las personas que están en esta situación; las cardinalidades mínimas a ambos lados serían entonces uno, pues para estar casado, tienes que estar casado como mínimo con una persona. El hecho de que este ejemplo derive en un caso de reflexividad (1, 1) ↔ (1, 1) nos da idea de que las parejas son listas circulares también, pero en este caso son círculos de dos nodos.

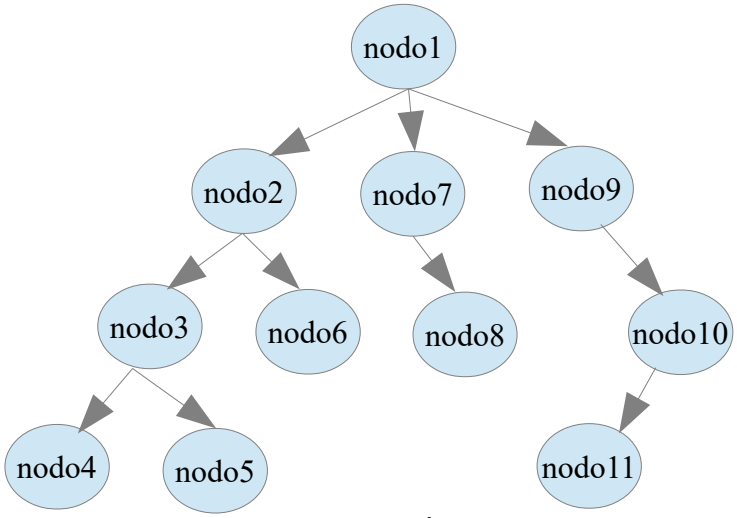


Figura 50: Estructura de datos de árbol.

6.1.2 Árboles utilizando reflexivas 1:N

Un árbol es una estructura de datos en la que cada nodo recibe como máximo una flecha, pero del que pueden partir varias. En un árbol siempre hay un único nodo *raíz*, que es el que no recibe ninguna flecha (el *nodo1* en la Figura 50), y puede haber múltiples nodos de los que no salgan flechas, que se llaman *hojas* (los nodos 4, 5, 6, 8 y 11 son hojas en la figura).

Los árboles sirven para representar jerarquías, que son muy comunes en la vida real. Por ejemplo, un árbol podría representar quién es jefe de quién en una empresa, qué partes constituyen el despiece de un mecanismo, y como esas partes se dividen en subpartes, etc …

En la Figura 43, ya habíamos representado la interrelación *se jefe de*, que es la interrelación que viene en todos los libros para ilustrar este caso.

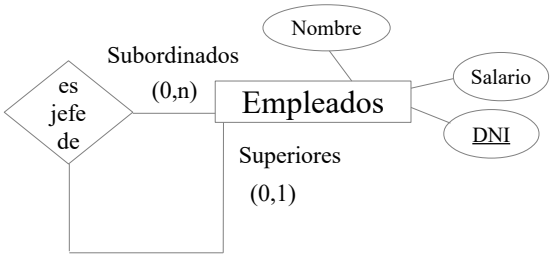


Figura 51: Ejemplo de interrelación reflexiva 1:N.

Volviendo al mismo ejemplo, nos fijamos en las cardinalidades. A un empleado le corresponde un sólo jefe directo como máximo, pero puede que cero si se trata del jefe máximo (i.e., la *raíz* del árbol), y por eso pusimos (0, 1) en el lado *superiores*. A un empleado le pueden corresponder varios subordinados directos en general, pero en algún caso, el empleado puede que no mande sobre nadie (i.e., en ese caso sería una *hoja*); y por ello pusimos (0, n) en el lado *subordinados*.

Por tanto, las cardinalidades mínimas cero están dando representación a la raíz y a las hojas del árbol. Sin embargo, esas dos cardinalidades hay quién las puede sustituir por "1" a través de algún convenio más o menos artificial, típicamente:

1. Podemos poner (1, 1) en el lado *superiores* si admitimos que el jefe supremo es jefe de si mismo, y por tanto en ese caso, hasta el jefe supremo tendría como mínimo y como máximo un jefe.
2. Podemos poner (1, n) en el lado *subordinados* si admitimos que los empleados que no tienen a nadie por debajo, tienen como subordinados a ellos mismos, y por tanto en ese caso, hasta el empleado más abajo del escalafón tendría como mínimo y un empleado a su cargo.

Esta forma opcional de hacer las cosas no es incorrecta, pero puede resultar confusa.

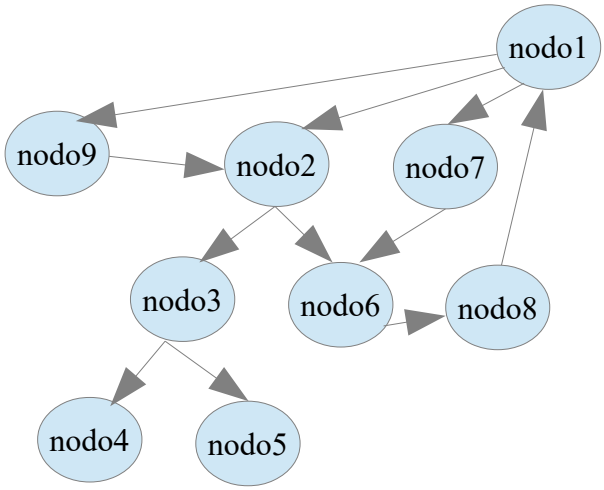


Figura 52: Estructura de datos de grafo.

6.1.3 Grafos dirigidos utilizando reflexivas N:M

En un grafo dirigido cada nodo puede recibir *n* flechas, y puede a su vez ser origen de *m* flechas. Se dice que es dirigido precisamente porque usamos flechas, si utilizásemos líneas que no fuesen flechas para unir los nodos, los grafos no serían dirigidos. Un grafo dirigido puede ser cíclico si existe al menos un nodo, para el que existe al menos un camino que partiendo de ese nodo y siguiendo las flechas, le lleve de vuelta a ese nodo. En caso contrario el grafo es acíclico.

En la Figura 52 vemos un grafo, se ve rápidamente que es un grafo y no un árbol, por que por ejemplo el *nodo 2* recibe más de un arco. En este caso el grafo es cíclico porque, por ejemplo, podríamos salir del *nodo 1* y regresar a el

después de recorrer, por ejemplo, el 7, el 6 y el 8; pero hay más ciclos en la figura, puedes intentar descubrilos como pasatiempos.

En la vida real, hay muchos objetos y escenarios que se pueden representar mediante grafos. Por ejemplo, un mapa de carreteras es un grafo que conecta ciudades, aunque en ese caso no es dirigido; pero si que es dirigido el mapa de las rutas de los coches de línea, de los autocares o de los aviones, porque puede que exista una conexión directa que nos lleve de la ciudad *A* a la *B*, pero eso no implica que exista la conexión inversa de la *B* a la *A*. También hay grafos cuando se representa un plan de trabajo y cada nodo es una tarea, cuando acabamos una tarea vamos a la siguiente, pero varias tareas pueden confluir en la misma tarea siguiente. Otro montón de ejemplos viene del mundo de la programación; por ejemplo un método o un procedimiento puede invocar a *n* y puede ser invocado por *m*, etc …

En la Figura 44 veíamos un ejemplo de ciudades conectadas por una compañía de autocares. Las cardinalidades se eligieron (1, n) en ambos extremos, porque quizás pensamos que toda ciudad puede ser origen y destino de múltiples líneas. Eso efectivamente justifica la cardinalidad máxima *n* en ambos extremos; sin embargo podemos tener dudas sobre la cardinalidad mínima:

1. ¿Tiene sentido que tengamos en la base de datos una ciudad a la que no le llega ninguna línea de autocares?. Podríamos pensar en una ciudad a la que no le llega ninguna línea, pero que sirve de origen a otras, y en ese caso la cardinalidad mínima en el lado *destino* es cero. Pero puede que pensemos que es una situación poco realista, porque los autocares que parten de esa ciudad, primero habrán tenido que llegar de alguna parte, y en ese caso la cardinalidad mínima del lado *destino* debiera de ser 1.
2. ¿Tiene sentido que tengamos en la base de datos una ciudad de la que no salgan autocares?. Podríamos pensar en una ciudad de la que no salen autocares de ninguna línea, pero que sirve de destino a otras, y en ese caso la cardinalidad mínima en el lado *origen* es cero. Pero puede que pensemos que sea una situación poco realista, porque los autocares que llegan a esa ciudad es de suponer que en algún momento salgan de ella, y en ese caso la cardinalidad mínima del lado *destino* debiera de ser 1.

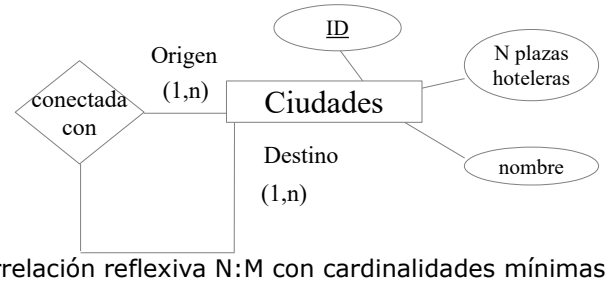


Figura 53: Interrelación reflexiva N:M con cardinalidades mínimas 1.

Carnalidad mínima “0” ó “1” son totalmente válidas apriori, depende de los requisitos del problema. Sustituyamos las ciudades por paradas de autobús en el ejemplo anterior.

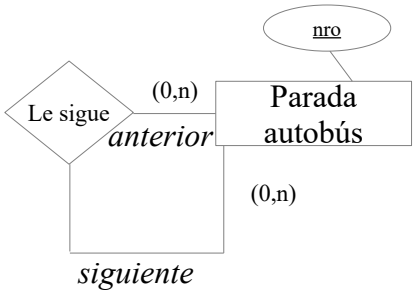


Figura 54:Interrelación reflexiva N:M con cardinalidades mínimas 0.

Una parada sí puede ser origen y nunca destino, si pertenece exclusivamente a líneas que no son circulares; y en la misma circunstancia, una parada puede ser destino y nunca origen. Esto es debido a que el autobús al acabar de

recorrer una línea no circular, puede empezar a dar servicio a otra línea desde la primera parada sin hacer grandes desplazamientos, mientras que para los autocares del ejemplo anterior sería ruinoso desplazarse de una ciudad a otra para comenzar a dar servicio a otra línea sin llevar pasajeros.
En el ejemplo de los autocares la cardinalidad mínima más adecuada parece "1" , en el de los autobuses, parece que es "0". Por tanto, en el caso N:M no está tan claro cuál será la cardinalidad mínima, dependerá de cada problema.
Como ocurría con las interrelaciones 1:1, las situaciones cíclicas o circulares, son las que nos llevan a cardinalidades mínimas "1", las situaciones lineales o abiertas, son las que nos llevan a cardinalidades mínimas "0".

6.2. Pasar a tablas las interrelaciones reflexivas

Las reglas de paso a tablas relaciones de las interrelaciones reflexivas son las mismas que cuando no son reflexivas. Simplemente hay que adaptarlas aplicando el sentido común.

6.2.1 Casos uno a varios

Empezaremos por este caso por ser más simple que el uno a uno. Tomemos el ejemplo anterior de *ser jefe de* (Figura 43). Como es una interrelación uno a muchos no genera una tabla adicional, luego se hace todo con una única tabla de empleados. La tabla tendrá los atributos propios de la entidad, el AIP como clave primaria, pero además, como es extremo "a varios" de la interrelación, incluirá el AIP otra vez como clave ajena a si mismo. Luego **el AIP hay que incluirle <u>dos</u> veces**, una como clave primaria y otra como ajena a si mismo, tal que así:

```
CREATE TABLE empleados(
     DNI        NUMERIC(8) PRIMARY KEY,
     DNI_Jefe   NUMERIC(8) REFERENCES empleados,
     nombre     VARCHAR(30) NOT NULL,
     salario    NUMERIC(7, 2)
);
```

El campo *DNI_Jefe* no le hemos puesto NOT NULL porque la cardinalidad del extremo "a uno" es (0,1), no (1,1), como se discutió ya en la sección 6.1.2. Esta cardinalidad tiene sentido pensando en que puede haber alguien que no tenga jefe, por ejemplo el jefe supremo, o quizás los múltiples miembros de un hipotético consejo de administración.
Por ejemplo, supongamos la siguiente jerarquía:

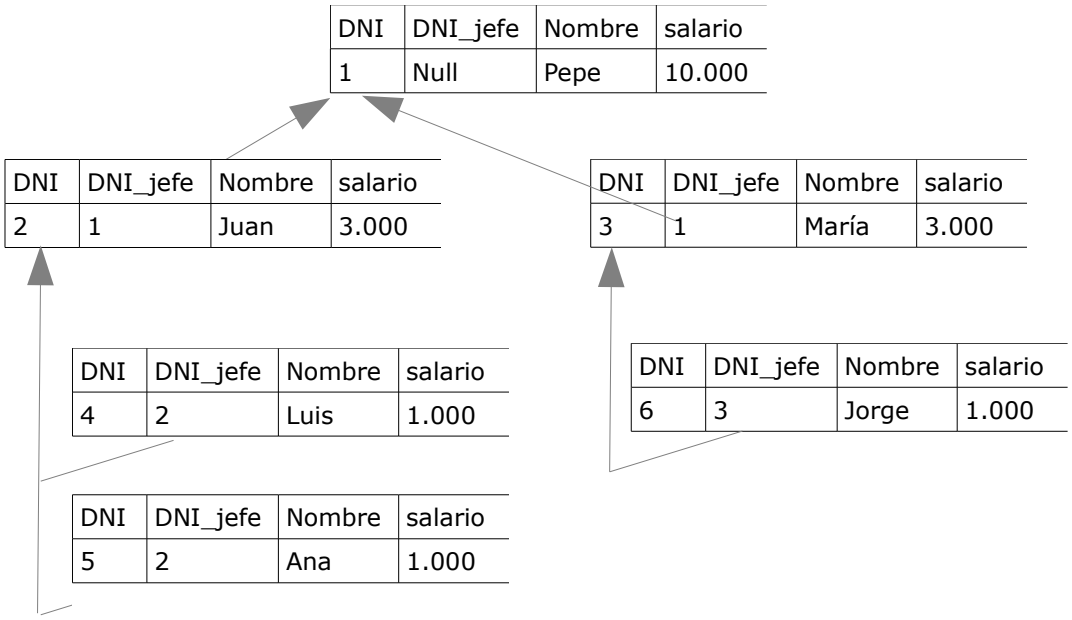


Figura 55: Jerarquía de mando representada en la interrelación "es jefe de".

que da lugar a la correspondiente tabla de la Figura 56, en la que el jefe supremo *Pepe*, como no tiene jefe en *DNI_Jefe* tiene el valor null.

Nota que no podemos insertar las filas en cualquier orden, siempre hay que insertar al jefe antes que al subordinado. En la sección 6.2.2 verás alguna idea que permitiría saltarse ese orden de inserción.

DNI	**DNI_jefe**	**Nombre**	**salario**
1	Null	Pepe	10.000
2	1	Juan	3.000
3	1	María	3.000
4	2	Luis	1.000
5	2	Ana	1.000
6	3	Jorge	1.000

Figura 56: Estructura jerárquica de la Figura 55 representada como una tabla relacional.

Como comentamos en la sección 6.1.2, aún siendo artificioso, también se podía haber pensado desde el principio que el extremo "a uno" tuviese una cardinalidad (1,1) y que en ese caso el jefe del jefe supremo fuese el mismo. En ese caso, el campo *DNI_jefe* de la tabla había que haberlo declarado como NOT NULL, y el valor de *DNI_jefe* sería en todas las filas igual que en la Figura 56, con excepción de la fila de *Pepe*, en la que *DNI_jefe* sería "1".

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

***Ejercicio**: Haz un diagrama E/R genérico correspondiente a una interrelación reflexiva 1:N. Utiliza letras para designar los nombres de la entidad, la relación y los atributos que necesites. Deduce las dependencias funcionales y las claves. ¿A que forma normal llegas y por qué?.*

6.2.2 Caso uno a uno

El caso 1:1, sería idéntico. Por ejemplo, las sagas cinematográficas de la Figura 47, se implementarían igual que los

empleados de la sección anterior (recuerda que el caso 1:1 siempre le hemos considerado una caso especial de 1:N), pero con la diferencia habitual de las 1:1; esto es: la clave ajena ha de ser UNIQUE.

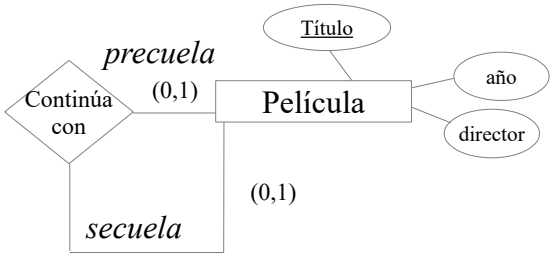


Figura 57: Interrelación reflexiva 1:1 representando una lista enlazada abierta.

```
CREATE TABLE película(
     título          VARCHAR(40) PRIMARY KEY,
     le_sigue        VARCHAR(40) REFERENCES película
                        UNIQUE,
     año             SMALLINT,
     director        CHAR(50)
);
```

El UNIQUE en este caso, evita que la misma película sea continuación de dos películas distintas.
Nota nuevamente que no puedes insertar las filas en cualquier orden; primero hay que insertar *Rocky I* y luego *Rocky II* para poder rellenar el campo *le_sigue* con el valor *Rocky I* sin violar la clave ajena. Un poco más adelante volveremos sobre esta misma cuestión con el ejemplo de los *vértices de las manzanas*.
Como se discutió en la sección 6.1.1, no parece adecuado tratar este caso con cardinalidades mínimas "1". Un caso con cardinalidades mínima "1" es de la Figura 48.

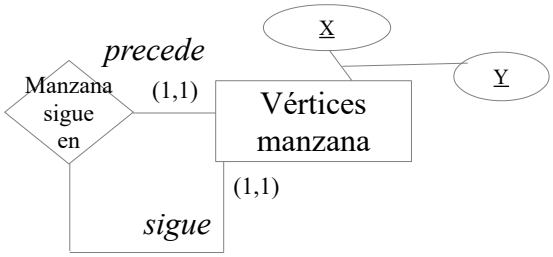


Figura 58: Interrelación reflexiva 1:1 representando una lista enlazada circular.

que daría lugar a una tabla como esta, en la que hemos hecho NOT NULL a los campos miembros de la clave ajena para preservar la cardinalidad mínima 1 (i.e., es obligatorio que cada vértice tenga un vértice siguiente).

```
CREATE TABLE vértices_manzanas(
     X              FLOAT,
     Y              FLOAT,
     X_siguiente    FLOAT NOT NULL,
     Y_siguiente    FLOAT NOT NULL,
      PRIMARY KEY ( X, Y),
     CONSTRAINT FK_vm
      FOREIGN KEY ( X_siguiente, Y_siguiente)
           REFERENCES vértices_manzanas DEFERRABLE,
      UNIQUE ( X_siguiente, Y_siguiente)
);
```

Podíamos, en principio, no haber dado nombre a ninguna restricción ni haber puesto la palabra DEFERRABLE, la razón por la que le hemos dado nombre a la clave ajena y hemos puesto DEFERRABLE la entenderás a continuación. Supongamos que no hacemos la clave ajena DEFERRABLE.. El CREATE TABLE, apriori, sería correcto, aunque es problemático a la hora de insertar puntos. Supongamos que vamos a insertar una manzana formada por 3 puntos { (0,0), (1,1), (1,0) }. Si insertamos primero el (0,0), ¿ cómo podríamos no violar el la clave ajena y/o el NOT NULL si aún no se ha insertado el (1, 1) ?. No es sencillo:

1. Una solución que no funciona es insertar cada punto preservando el carácter cerrado del polígono de la manzana; por ejemplo así:
   Inserto el primer punto como siguiente de si mismo

   ```
   INSERT INTO vértices_manzanas VALUES ( 0, 0, 0, 0);
   ```

   Inserto el segundo punto como siguiente del primero, y al primero lo dejo como anterior del segundo mediante un INSERT y un UPDATE como los siguientes:

   ```
   INSERT INTO vértices_manzanas VALUES ( 1, 1, 0, 0);
   UPDATE  vértices_manzanas
   SET X_siguiente = 1, Y_siguiente = 1
   WHERE X=0 AND Y=0;
   ```

   Pero si ejecuto primero el insert se viola el UNIQUE, y si por el contrario ejecuto primero el UPDATE se violaría la clave ajena.

2. Otra posibilidad, no utilziando la declaración DEFERRABLE sería insertar cada manzana con una sola sentencia INSERT. Por ejemplo:

   ```
   INSERT INTO vértices_manzanas VALUES ( 0, 0, 0, 0),
                                        ( 1, 1, 1, 1),
                                        ( 1, 0, 1, 0);
   ```

como las restricciones se comprueban al final de la ejecución de cada sentencia INSERT/UPDATE/DELETE, y hay una sola sentencia INSERT que inserta los 3 puntos a la vez, no hay un estado intermedio en el que la comprobación provoque algún tipo de violación. Sin embargo, esta solución no funciona en todos los SGBDs (p.e. Oracle no admite inserciones multifila).

3. Una posibilidad es declarar las restricción de clave ajena como *diferidas*. Es por ello, que hemos dado nombre a esa restricción y la hemos declarado como *diferible* (i.e., DEFERRABLE). Al hacerla diferible ganamos que la comprobación de si se viola la clave ajena se haga más adelante. Ahora podemos hacer:

```
BEGIN TRANSACTION;

SET CONSTRAINTS FK_VM DEFERRED;

INSERT INTO vértices_manzanas VALUES ( 0, 0, 0, 0);
INSERT INTO vértices_manzanas VALUES ( 1, 1, 1, 1);
INSERT INTO vértices_manzanas VALUES ( 1, 0, 1, 0);

COMMIT;
```

que en definitiva consiste en establecer dos instantes en la operación, BEGIN TRANSACTION y COMMIT, que indican respectivamente cuando empieza y cuando finaliza. Durante ese intervalo hemos diferido la clave ajena con el SET CONSTRAINTS, con lo que hemos podido meter los 3 vértices en cualquier orden, pues no se va a comprobar la clave ajena hasta llegar al último COMMIT.
La operación COMMIT es muy importante y tiene otras muchas implicaciones muy importantes que estudiarás el próximo curso en la asignatura de *Aplicaciones de Bases de Datos.*

4. Finalmente, la posibilidad más sencilla es volver a quitar los NOT NULL y proceder en dos fases:
   1. Insertar los 3 puntos de manera que el punto siguiente siempre sea nulo, y
   2. Actualizar más tarde quien es el siguiente de cada uno, controlando nosotros mismos que no queden polígonos abiertos.

   Si la inserción de puntos en el polígono se hace desde un programa en *C++ java*, etc... sería este programa el que llevaría a cabo este control.

Discutamos ahora el caso de las personas casadas (Figura 49).

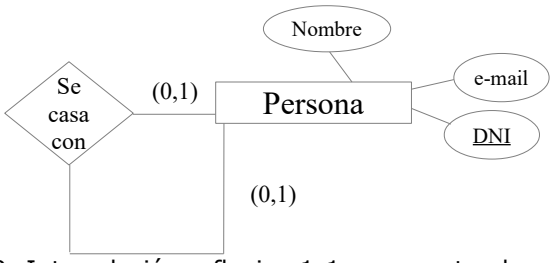


Figura 59: Interrelación reflexiva 1:1 representando parejas.

Que tras todo lo visto da lugar a:

```
CREATE TABLE personas(
        DNI             NUMERIC(8) PRIMARY KEY,
        DNI_conyuge     NUMERIC(8) REFERENCES personas
                           UNIQUE,
        nombre          CHAR(30) NOT NULL,
        email           CHAR(50)
);
```

Lógicamente la poligamia está prohibida por la restricción UNIQUE, que evita que el DNI del cónyuge aparezca varias veces como tal en la tabla.

Por ejemplo, supongamos el contenido de la tabla correspondiente a la Figura 60, en la que *Pepe* está casado con *María*, y *Juan* está soltero, valiendo su cónyuge *null*. Las cardinalidades mínimas eran cero precisamente para permitir solteros, viudos etc...

DNI	DNI_conyuge	Nombre	email
1	2	Pepe	pepe@pepe.com
2	1	María	maria@maria.com
3	Null	Juan	juan@juan.com

Figura 60: Contenido de una tabla de ejemplo representando una interrelación 1:1.

Date cuenta de que la tabla de ejemplo no impide triángulos amorosos, ni otro tipo de polígonos polígamos, ya que *Pepe* podría tener como DNI de cónyuge el de *María*, *María* el de *Juan*, y *Juan* el de *Pepe*, y no se violaría ninguna de las restricciones declaradas en el CREATE TABLE. Incluso *Pepe* podría estar casado con *María*, y ésta tener un cónyuge *null.* Todo esto no tiene una solución sencilla.

1. Una posibilidad para solucionarlo es utilizar un sistema de *triggers*. Los *triggers* se ven en la asignatura de tercero *Aplicaciones de Bases de Datos*. Podríamos tener un *trigger* que cuando A actualiza el DNI de su cónyuge al DNI de B, comprobase que B está soloter@ y en ese caso le asignase automáticamente el DNI de A como DNI de cónyuge.
2. Otra posibilidad es usar ASSERTIONs[11], pero los SGBDs rara vez las soportan. Una ASSERTION es un CHECK que se declara aparte – fuera del CREATE TABLE – que controla cosas como por ejemplo: si el resultado de una SELECT es el esperado. Podríamos declarar una ASSERTION con una SELECT que comprobase que no hay situaciones ilegales en los matrimonios.
3. Las posibilidades 1 y 2 consumen muchos recursos, imagínate que cada vez que haces una inserción se dispara una SELECT más o menos compleja haciendo todas esas comprobaciones, y por ello normalmente sobrecargan al SGBD y pueden degradar su rendimiento. Por ello, lo habitual es que si los matrimonios se introducen desde un programa *C++, java* etc... sea el propio programa el que controle que no se producen estas anomalías como los "tríos matrimoniales".

   Nota: Pero esto tampoco es una solución ideal, ya que cada vez que se haga un nuevo programa que maneje en estos datos tendré que volver recordar que he de comprobar primero si el cónyuge está casado, etc .. Por el contrario, en las soluciones anteriores, la base de datos hace esa comprobación siempre, descargando a

11 En el Glosario parece una explicación detallada de lo que es uno objeto ASSERTION, con el correspondiente ejemplo.

los programadores.

Si, como se discutió al final de la sección 6.1.1, hacemos que la entidad *Personas* pase a ser *Personas-Casadas*, las cardinalidades de ambos extremos serían 1. Esto evitaría el caso *"Pepe podría estar casado con María, y ésta tener un cónyuge null.",* pero no evita el problema de los "tríos matrimoniales", que nuevamente debería de ser controlado con cualquiera de las 3 opciones que acabamos de ver (i.e., *triggers, ASSERTION,* o controlarlo desde un programa de aplicación).

Lógicamente, si optamos por (1, 1) en ambos extremos de la interrelación *se casa con,* tendríamos los mismos problemas que con *manzana sigue en*: si voy a casar a *Pepe* con *María*, no puedo insertar a *Pepe* hasta que no inserte primero a *María*, y viceversa, porque si inserto primero a *Pepe* al poner en *DNI_conyuge* el DNI de *María* se violaría la clave ajena. Nuevamente, las soluciones pasan o por los *triggers*, o por las restricciones diferibles, o por quitar los NOT NULLs y que sea el propio programa de aplicación el que controle el caso *"Pepe podría estar casado con María, y ésta tener un cónyuge null."*.

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

***Ejercicio**: Haz un diagrama E/R genérico correspondiente a una interrelación reflexiva 1:1. Utiliza letras para designar los nombres de la entidad, la relación y los atributos que necesites. Deduce las dependencias funcionales y las claves. ¿A que forma normal llegas y por qué?.*

6.2.3 Caso varios a varios

Como en el caso N:M genérico, en el caso N:M reflexivo se genera una tabla nueva por la interrelación. Esta tabla incluiría dos veces la clave primaria de la entidad, una por cada extremo o rol de la interrelación; y como siempre en toda interelación N:M, la clave primaria de esta nueva tabla estaría formada por las dos claves ajenas. Las claves ajenas normalmente llevan restricciones en cascada. En el ejemplo de la Figura 53:

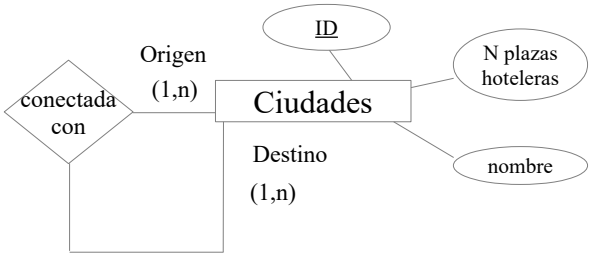


Figura 61: Interrelación reflexiva N:M con cardinalidades mínimas 1.

```
CREATE TABLE ciudades(
     ID        INTEGER PRIMARY KEY,
     nombre          CHAR(20) UNIQUE NOT NULL,
     n_plazas  INTEGER
);
```

```
CREATE TABLE conecta(
     idOrigen  INTEGER REFERENCES ciudades
               ON DELETE CASCADE ON UPDATE CASCADE,
     idDestino INTEGER REFERENCES ciudades
               ON DELETE CASCADE ON UPDATE CASCADE,
     PRIMARY KEY( idOrigen, idDestino)
);
```

Nota que perdemos el carácter ***dirigido*** del grafo (i.e., ya no hay flechas, o si se quiere las flechas tendrían puntas en los dos sentidos), a diferencia de las listas enlazadas en los que la clave ajena nos llevaba siempre al siguiente nodo, y a diferencia de los árboles en los que la clave ajena nos llevaba siempre al nodo padre, en los grafos representados mediante una tabla tenemos dos claves ajenas, y eso nos permite navegar en los dos sentidos. Esa misma idea, permite intuir que si quisiéramos implementar para una interrelación 1:1 una lista bidireccionel (i.e., lo que se conoce como una lista doblemente enlazada), deberíamos de sacar la interrelación 1:1 a una tabla con dos claves ajenas como se hizo en la sección 4.4.3.

Debido a que desde un CREATE TABLES de SQL no podemos verificar si se viola el "1" de una restricción (1, n), esta regla la aplicaremos igual tanto si las cardinalidades son (0, n) como si son (1, n). Si realmente se quiere controlar el "1" de (1, n) hay que recurrir bien a *triggers*, bien a una ASSERTION[12], o típicamente que sea el propio programa de aplicación el que ejerza ese control.

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

***Ejercicio**: Haz un diagrama E/R genérico correspondiente a una interrelación reflexiva N:M. Utiliza letras para designar los nombres de la entidad, la relación y los atributos que necesites. Deduce las dependencias funcionales y las claves. ¿A que forma normal llegas y por qué?.*

7. Las Interrelaciones de grado superior a dos

7.1. Justificación de las interrelaciones de grado superior

Queremos modelar el plan de ordenación docente donde consta el número de horas que cada *profesor* da a cada *grupo* de cada *asignatura*. Supongamos los AIPs *IdProfesor, IdGrupo* e *IdAsignatura* respectivamente para cada una de las tres entidades. Veamos varias alternativas, sus pros y contras.

Aproximación 1: Modelar los Grupos como atributo de una binaria

Una primera aproximación al problema podía habernos llevado a entenderlo como una interrelación binaria N:M:

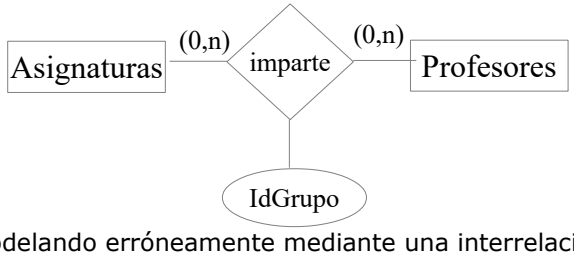


Figura 62: Modelando erróneamente mediante una interrelación binaria.

12 En el Glosario parece una explicación detalla de lo que es uno objeto ASSERTION, con el correspondiente ejemplo.

Pero entonces recordemos que *imparte* es al final un conjunto de "flechas" que van de asignaturas a profesores, y por tanto no puede haber flechas repetidas, es decir, un profesor no podría relacionarse más de una vez con la misma asignatura. La consecuencia directa es que un profesor sólo podría como máximo impartir un grupo de cada asignatura, por lo cual es erróneo pensar en una binaria, tal cual.

Se puede ver incluso de otra manera: si *imparte* fuera binaria N:M, generaría una tabla cuya clave primaria sería la compuesta *(idAsignatura, idProfesor)*, es decir: nuevamente estaríamos prohibiendo que la combinación *Profesor-Asignatura* se repitiese, y que por tanto un profesor pudiera impartir varios grupos de la misma asignatura. Nuestro diseño debería de permitir que un profesor impartiera varios grupos de la misma asignatura.

Aproximación 2: Reificar[13] la interrelación

Otra solución consistiría en convertir la relación en entidad:

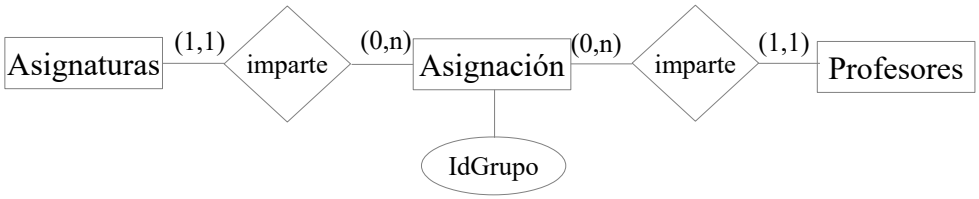


Figura 63: Reificación de la interrelación.

La relación imparte se convierte en entidad, que algunos autores la llaman *entidad-asociación* (*connecting entity* en [Ullman y Widom 1997] sección 2.2.5). Esta representación si que permite que un profesor imparta varias veces la misma asignatura, supuestamente a grupos distintos.

Sin embargo, nosotros sabemos que cada combinación *asignatura-profesor-grupo* no se debería de poder repetir (i.e., es una clave candidata de la tabla *Asignación*), mientras que las tablas que salen del diagrama de la Figura 63, si que lo permitiría.

Además este diagrama no permite representar otras situaciones. Por ejemplo, supongamos que dada una asignatura y un grupo sólo lo pudiera impartir como máximo un profesor. El diagrama anterior es claramente incapaz de representar esa restricción, tendríamos que ampliarlo con comentarios o anotaciones.

Aproximación 3: Utilizar una interrelación de grado superior (i.e. una ternaria)

La solución que resolvería los problemas de la aproximación anterior es utilizar una interrelación ternaria o de grado 3:

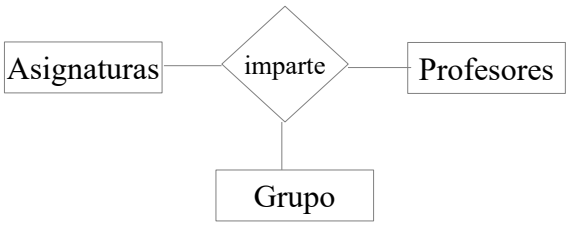


Figura 64: Representar *asignaturas-grupos-profesores* mediante una interrelación ternaria.

Como veremos en las siguientes secciones, podremos utilizar las cardinalidades para representar restricciones del tipo "*dada una asignatura y un grupo sólo lo pudiera impartir como máximo un profesor*", y además existirán reglas de paso de estas relaciones a tabla por las que obtendremos tablas que recojan esas restricciones.

7.2. Las cardinalidades de Chen en las ternarias

Las cardinalidades para interrelaciones de grado *n* superior a dos que aparecen en los libros de texto, son normalmente la de Chen [Chen 1976], [McAllister 1995]. Estas cardinalidades se toman fijando una combinación de

13 Según el diccionario de la RAE, *reificar* es cosificar o convertir algo en cosa.

valores cualquiera pertenecientes a las otras *n-1* entidades, y viendo con cuantas de ellas se **puede** relacionar la entidad cuyo arco estemos analizando en ese momento como mínimo y como máximo. Por ejemplo:

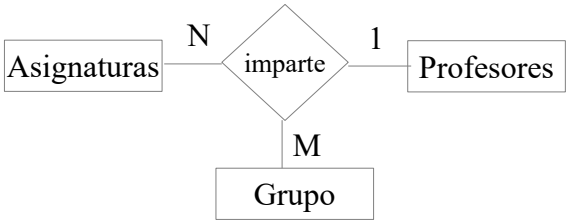


Figura 65: Análisis de cardinalidades en una interrelación ternaria.

1. Para ver la cardinalidad en el **lado de asignaturas** elegimos un par cualquiera de *profesor* y *grupo*. Por ejemplo, el profesor *P* y el grupo *G*.
    1. El profesor *P* puede que no de docencia al grupo *G*, por lo tanto ese par, en caso de no existir, podría relacionarse cero veces con una asignatura, luego la cardinalidad mínima es cero.
    2. El profesor *P* puede que imparta al grupo *G* en varias asignaturas, luego la cardinalidad máxima es N.
2. Para ver la cardinalidad en el **lado de profesores** elegimos un par cualquiera de *asignatura* y *grupo.* Por ejemplo, la asignatura *A* y el grupo *G*.
    1. La asignatura *A* puede que no tenga un grupo *G* (por ejemplo, si ese grupo es de repetidores y esa asignatura tiene un número muy pequeño de repetidores), por lo tanto ese par, en caso de no existir, le corresponderían cero profesores, luego la cardinalidad mínima es cero. Otra forma de razonarlo: Como los grupos son distintos por curso (no es lo mismo el grupo 1 de primero que el grupo 1 de segundo), si el par que elijo lo forman una asignatura de primero con un grupo de segundo, daría lugar a una combinación *asignatura-grupo* inexistente, a la que le corresponderán cero profesores.
    2. El grupo *G* de una asignatura *A*, vamos a suponer que en los requisitos nos dicen que sólo puede ser impartido como máximo por un profesor. Pues bien, en ese caso la cardinalidad máxima es uno.
3. Para ver la cardinalidad en el **lado de grupo** elegimos un profesor *P* cualquiera y una asignatura *A* cualquiera.
    1. Ese profesor puede que no imparta esa asignatura, luego el par (*P, A*) podría no existir, y por tanto, nunca se relacionaría con un grupo, luego la cardinalidad mínima es cero.
    2. *P* puede impartir varios grupos de *A*, luego la máxima es N.

La cardinalidad mínima de Chen es siempre cero, de lo contrario cualquier par que cogiésemos de dos entidades se relacionaría **siempre** con la tercera, y eso no ocurre en la práctica, por ejemplo:

- Para que el lado *asignaturas* hubiera tenido cardinalidad mínima uno, a todo par {*profesor, grupo*} le correspondería siempre al menos una asignaturas. Eso significa que en la interrelación ternaria existiría al menos una fila para toda combinación posible de *profesor* y *grupo.* Por tanto estaríamos exigiendo que:
    - Todo profesor imparta (se relaciona) en todos los grupos y viceversa, o lo que es lo mismo:
    - Todo grupo es impartido por todos los profesores y todo profesor imparte todos los grupos
- Para que el lado *grupos* hubiera tenido cardinalidad mínima uno, a todo par {*profesor, asignatura*} le correspondería siempre al menos un grupo. Eso significa que en la interrelación ternaria existiría al menos una fila para toda combinación posible de *profesor* y *asignatura.* Por tanto estaríamos exigiendo que:
    - Todo profesor imparte (se relaciona) con todas las asignaturas y viceversa, o lo que es lo mismo:
    - Toda asignatura es impartida por todos los profesores y todo profesor imparte todas las asignaturas
- Y para que el lado *profesores* hubiera tenido cardinalidad mínima uno, a todo par {*grupo* , *asignatura*} le correspondería siempre al menos un profesor. Eso significa que en la interrelación ternaria existiría al

menos una fila para toda combinación posible de grupo y asignatura. Por tanto estaríamos exigiendo que:

- Se impartiese cada grupo en en todas las asignaturas y viceversa, o lo que es lo mismo:
- Toda asignatura se imparta en todos los grupos, y todo grupo es impartido en todas las asignaturas.

Vemos que normalmente todos estos supuestos son muy restrictivos. El caso del lado de profesores, quizás es el único al que podríamos dar algo de crédito, quizás en una institución en la que todas las asignaturas tuviesen el mismo número de grupos (i.e. aproximadamente el mismo número de alumnos en todas las asignaturas), y en una titulación de un sólo curso, porque si tuviera más de un curso ya no existirían las combinaciones de, por ejemplo, las asignaturas de primero con los grupos de segundo. Pero con todo, no deja de ser una situación inusual. Por ello, no representaremos las cardinalidades mínimas (como en la figura) pues en la práctica siempre son cero.

7.2.1 El caso N:M:P y su representación en tablas

En este caso las tres patas del diagrama tienen cardinalidad Chen "a muchos". La regla que aplicamos es muy similar a la de las relaciones binarias N:M. Para pasar a tablas una interrelación de grado *n:*

1. Cada una de las *n* entidades extremo da lugar a una tabla cuyos atributos se deducen de las reglas vistas hasta ahora.
2. La relación da lugar a una tabla adicional
    1. Cuyos atributos son los *n* AIPs de las entidades que conecta más los atributos propios de la interrelación .
    2. Que mantiene *n* claves ajenas, una con cada tabla correspondiente a las entidades extremo de la interrelación. Cada clave ajena se forma con los campos de uno de los AIPs de esas entidades. Estas claves ajenas es usual, pero no imprescindible, que implementen ON DELETE CASCADE y ON UPDATE CASCADE.
    3. Cuya clave primaria es compuesta, y está formada por las *n* claves primarias de las *n* tablas que relaciona.

Supongamos el mismo ejemplo que antes, pero permitiendo que varios profesores puedan intervenir en una mismo grupo de una asignatura, entonces se nos transforma en un M:N:P.

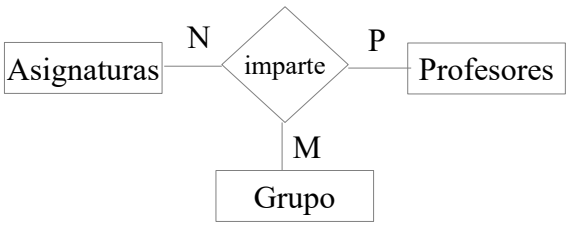


Figura 66: Ternaria N:M:P.

En este caso, además de las tablas de asignaturas, profesores y grupos se crearía la tabla *imparte*:

```
CREATE TABLE imparte (
     idAsignatura   INTEGER REFERENCES asignaturas
          ON DELETE CASCADE ON UPDATE CASCADE,
     idProfesor     INTEGER REFERENCES profesores
          ON DELETE CASCADE ON UPDATE CASCADE,
     idGrupo        INTEGER REFERENCES grupos
          ON DELETE CASCADE ON UPDATE CASCADE,
     PRIMARY KEY (
          idAsignatura, idProfesor, idGrupo)
);
```

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

***Ejercicio**: Haz un diagrama E/R genérico correspondiente a una interrelación ternaria N:M:P. Utiliza letras para designar los nombres de la entidad, la relación y los atributos que necesites. Deduce las dependencias funcionales y las claves. ¿A que forma normal llegas y por qué?.*

7.2.2 El caso 1:M:N y su representación en tablas

Volvamos al ejemplo en el que no podría haber varios profesores impartiendo el mismo grupo de la misma asignatura

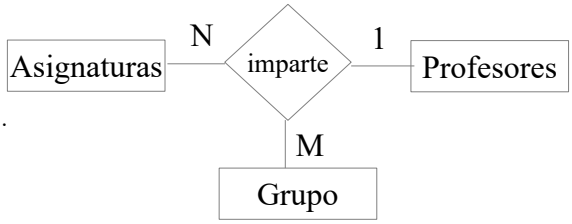


Figura 67: Ternaria 1:N:M.

Al aplicar la regla de transformación para N:M:P nos sale la misma tabla que en la sección anterior. Sin embargo, las cardinalidades nos indican que dada una asignatura y un grupo la corresponde como máximo un profesor. Es decir, una situación:

*(Asignatura_1, **Profesor_1**, Grupo_1)*

*(Asignatura_1, **Profesor_2**, Grupo_1)*

debería de estar totalmente prohibida, y sin embargo nuestra PRIMARY KEY la permite. Por ello, en este caso nos conviene más la clave primaria: ***IdAsignatura** + **IdGrupo***, de donde la tabla que se obtendría sería esta otra.

```
CREATE TABLE imparte (
     idAsignatura INTEGER REFERENCES asignaturas
          ON DELETE CASCADE ON UPDATE CASCADE,
     idProfesor INTEGER REFERENCES profesores
               ON DELETE CASCADE ON UPDATE CASCADE
          NOT NULL,
     idGrupo INTEGER REFERENCES grupos
          ON DELETE CASCADE ON UPDATE CASCADE,
     PRIMARY KEY ( idAsignatura, idGrupo )
);
```

(Hemos de añadir el NOT NULL de *idProfesor* porque ya no pertenece a la clave primaria, luego ya no está implícito que sea NOT NULL).

Regla: Por tanto, cuando uno de los extremos es uno, habrá una clave primaria compuesta por los AIPs de los otros dos extremos.

Ejercicio*: Comprueba en qué forma normal están las tablas obtenidas.*

Ejercicio*: Haz un diagrama E/R genérico correspondiente a una interrelación ternaria 1:N:M Utiliza letras para designar los nombres de la entidad, la relación y los atributos que necesites. Deduce las dependencias funcionales y las claves. ¿A que forma normal llegas y por qué?.*

7.2.3 El caso 1:1:N y su representación en tablas

Supongamos el siguiente ejemplo tomado de [Costal-Costa 2005]:

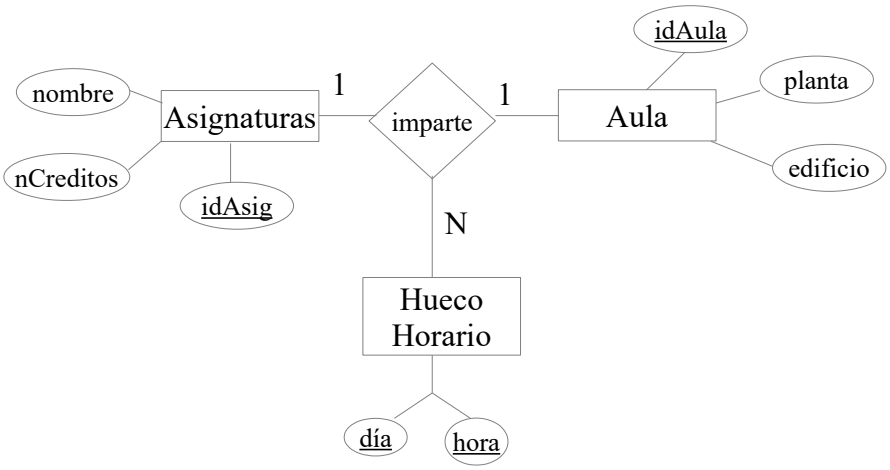


Figura 68: Ternaria 1:1:N.

Analicemos las cardinalidades máximas:

1. Lado *Asignaturas*: dado un aula y un hueco horario cualquiera, le corresponde como máximo una única asignatura, porque en un aula no puedo dar dos asignaturas al tiempo.
2. Lado *Aula*: dada una asignatura y un hueco horario cualquiera, le corresponde como máximo un único aula[14].

14 Podría darse el caso de que tuviésemos varios profesores de prácticas que imparten al mismo tiempo las prácticas de una asignatura, cada profesor en un aula distinta, pero vamos a suponer que no es así, ya que en ese caso el extremo aula podría ser

3. Lado *Huecos:* Dada una asignatura y un aula, puede ocupar varios huecos en el horario.

A la luz de las cardinalidades, nos damos cuenta que hay ciertas situaciones imposibles. Por ejemplo:

1. La cardinalidad 1 en el lado de asignaturas hace imposible:

   *(**Asignatura1**, Aula1, Lunes, 8:30)*

   *(**Asignatura2**, Aula1, Lunes, 8:30)*

2. La cardinalidad 1 en el lado aula hace imposible:

   *(Asignatura1, **Aula1**, Lunes, 8:30)*

   *(Asignatura1, **Aula2**, Lunes, 8:30)*

Sin embargo, si hacemos la transformación a relacional declarando la clave primaria como la combinación de los tres AIPs (caso N:M:P): (*idAsignatura, idAula, dia, hora*), las dos situaciones anteriores estarían permitidas. Luego por lo tanto esa clave primaria está mal y no podemos utilizar la regla de transformación de las n-arias N:M:P.

Para evitar el primer problema la combinación (*aula, dia, hora*) debiera ser clave candidata. Para evitar el segundo problema la combinación (*asignatura, dia, hora*) también debiera de ser clave candidata. Por tanto, tenemos dos claves candidatas y elegimos una cualquiera como primaria, y la otra la declaramos en SQL como UNIQUE con NOT NULL:

```
CREATE TABLE imparte (
     idAsignatura   INTEGER REFERENCES asignaturas
                    ON DELETE CASCADE
                    ON UPDATE CASCADE,
     idAula         INTEGER REFERENCES aulas
                    ON DELETE CASCADE
                    ON UPDATE CASCADE
                    NOT NULL,
     dia        CHAR(1)
                CHECK( dia IN ('L','M','X','J','V')),
     hora       TIME,
     FOREIGN KEY (dia, hora) REFERENCES huecos
          ON DELETE CASCADE ON UPDATE CASCADE,
     PRIMARY KEY ( idAsignatura, dia, hora),
     UNIQUE (idAula, dia, hora )
);
```

Regla: Por tanto, cuando dos de los extremos son uno, habrá dos claves candidatas compuesta cada una por el AIPs del extremos “a varios” y el AIPs de una de las partes “a uno”, y nuevamente el AIP del extremo “a varios ” y el AIPs de la otra parte “a uno”.

Ejercicio*: Comprueba en qué forma normal están las tablas obtenidas.*

Ejercicio*: Haz un diagrama E/R genérico correspondiente a una interrelación ternaria 1:1:N. Utiliza letras para*

N y ya no nos valdría el ejemplo.

designar los nombres de la entidad, la relación y los atributos que necesites. Deduce las dependencias funcionales y las claves. ¿A que forma normal llegas y por qué?.

7.2.4 El caso 1:1:1 y su representación en tablas

Veamos el siguiente ejemplo:

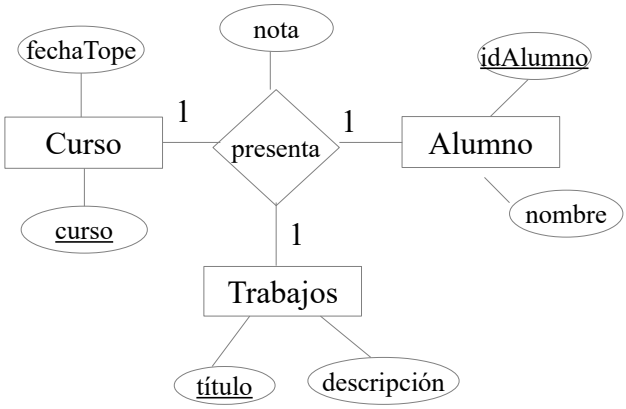


Figura 69: Ternaria 1:1:1.

Se trata de un profesor que para evaluar su asignatura tiene una lista de trabajos que más o menos es la misma en todos los cursos. Los trabajos son individuales (sólo un alumno por trabajo), y cada alumno presenta durante el curso sólo uno de los trabajos. El mismo trabajo sólo se puede presentar una vez en cada curso. Los alumnos repetidores no podrían realizar (repetir) el mismo trabajo en cursos diferentes. Estudiemos las cardinalidades:

1. Lado *alumnos*: dado un trabajo y un curso, sólo puede realizarlo como máximo un alumno.
2. Lado *curso*: dado un trabajo y un alumno, no puede repetirlo aunque sea repetidor, luego le corresponde un sólo curso como máximo.
3. Lado *trabajos*: dado un alumno y un curso, sólo tiene que hacer un trabajo.

Por tanto, se trata del caso 1:1:1. A la luz de estas cardinalidades, nos damos cuenta que hay ciertas situaciones imposibles. Por ejemplo:

1. La cardinalidad 1 en el lado de alumnos hace imposible:

   *(**Alumno1**, Trabajo1, 2008_9)*

   *(**Alumno2**, Trabajo1, 2008_9)*
2. La cardinalidad 1 en el lado curso hace imposible:

   *(Alumno1, Trabajo1, **2007_8**)*

   *(Alumno1, Trabajo1, **2008_9**)*
3. La cardinalidad 1 en el lado trabajos hace imposible:

   *(Alumno1, **Trabajo1**, 2008_9)*

   *(Alumno1, **Trabajo2**, 2008_9)*

Si utilizásemos la misma regla que para el caso M:N:P la clave primaria sería (*idAlumno, título, curso*), y podría haber filas que violasen cualquiera de los tres supuestos anteriores.

Para evitar el primer problema (*título, curso*) debería ser clave candidata, para evitar el segundo problema (*idAlumno, título*) debería ser clave candidata, y para evitar el tercer problema lo debería ser (*idAlumno, curso*). Elegiremos una cualquiera de estas claves como primarias, y las otras dos como UNIQUE asegurando que todos los campos de clave ajena sean NOT NULL, por ejemplo:

```
CREATE TABLE presenta (
     idAlumno       INTEGER REFERENCES alumnos
          ON DELETE CASCADE ON UPDATE CASCADE,
     curso          NUMERIC(4) REFERENCES cursos
          ON DELETE CASCADE ON UPDATE CASCADE,
     título         CHAR(10) NOT NULL REFERENCES trabajos
          ON DELETE CASCADE ON UPDATE CASCADE,
     nota      numeric(3,1),
     PRIMARY KEY ( idAlumno, curso),
     UNIQUE ( idAlumno, título),
     UNIQUE ( curso, título)
);
```

Regla: Por tanto, cuando los tres extremos son uno, habrá tres claves candidatas compuesta cada una por uno de llos 3 pares posibles de AIPs de los extremos.

Ejercicio*: Comprueba en qué forma normal están las tablas obtenidas.*

Ejercicio*: Haz un diagrama E/R genérico correspondiente a una interrelación ternaria 1:1:1. Utiliza letras para designar los nombres de la entidad, la relación y los atributos que necesites. Deduce las dependencias funcionales y las claves. ¿A que forma normal llegas y por qué?.*

7.3. Regla general para pasar a tablas una relación *n*-aria

Vistas las secciones anteriores, podemos generalizar la siguiente regla que es válida para ternarias, pero también para interrelaciones de grado superior a tres:

1. Cada una de las *n* entidades extremo da lugar a una tabla cuyos atributos se deducen de las reglas vistas hasta ahora.
2. La relación da lugar a una tabla adicional
    1. Cuyos atributos son los *n* AIPs de las entidades que conecta más los atributos propios de la interrelación .
    2. Que mantiene *n* claves ajenas, una con cada tabla correspondiente a las entidades extremo de la interrelación. Cada clave ajena se forma con los campos de uno de los AIPs de esas entidades. Estas claves ajenas es usual, pero no imprescindible, que implementen ON DELETE CASCADE y ON UPDATE CASCADE.
    3. Si todos los lados de la interrelación son “a varios”
        1. Tendrá una única clave primaria es compuesta, y está formada por las *n* claves primarias de las *n* tablas que relaciona.
    4. Sino
        1. Habrá una clave candidata por cada “partes a uno” de la interrelación.
            1. La clave correspondiente al extremo *i*-ésimo estará formada por los AIPs de las entidades del resto de los extremos

2. Elegiremos una de esas claves como primaria, y el resto como UNIQUE.

2. Si como consecuencia del paso anterior quedase algún atributo de alguna clave ajena que no perteneciese a la clave primaria, estos atributos se declararían NOT NULL.

7.4. Las interrelaciones ternarias como forma de modelado temporal en el Diagrama E/R

Supongamos que queremos modelar la matriculación de los alumnos en las asignaturas, contemplando el hecho de que un alumno podría haberse matriculado durante varios cursos de la misma asignatura.

Claramente es erróneo pensar en la binaria:

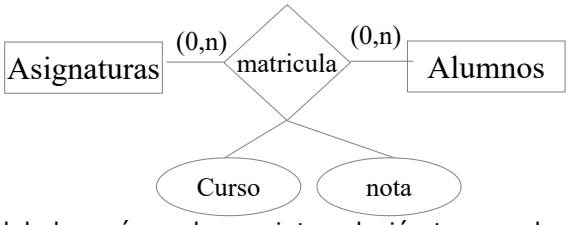


Figura 70: Modelado erróneo de una interrelación temporal como binaria.

pues al crear la tabla *matrícula* tendríamos una clave primaria *(idAsignatura, idAlumno*) que impediría que hubiese repetidores (el mismo alumno no podría relacionarse o matricularse varias veces de la misma asignatura).

Este problema es muy común. Tenemos una interrelación binaria N:M que es afectada por un atributo que añade una dimensión temporal o histórica, en este caso el *curso*. Esta situación se representa a través de una ternaria, en la que a la relación binaria la sale una tercera pata: una entidad que representa fechas o hitos temporales.

En nuestro ejemplo tendríamos la ternaria *Asignaturas-Alumnos-Cursos*. Otra posibilidad hubiera sido reificar, e inventarnos una entidad *Matrícula* que se relacione binariamente con las otras tres entidades, pero perderíamos la representación en el diagrama de la clave candidata compuesta *(idAsignatura, idAlumno, idCurso*) que impide que el mismo alumno se matricule en el mismo curso varias veces de la misma asignatura, y que sin embargo sí que está presente en la ternaria:

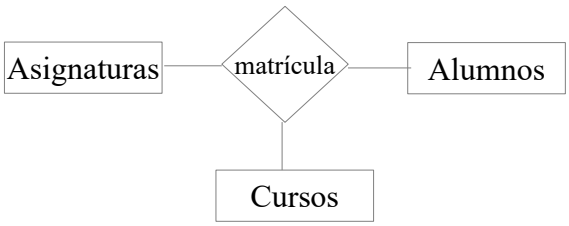


Figura 71: Modelado de la temporalidad mediante una ternaria.

Si analizáramos las cardinalidades de Chen obtendríamos que todas son N. En SQL:

```
CREATE TABLE asignaturas (
     idAsignatura   INTEGER PRIMARY KEY,
     nombre         CHAR(20) NOT NULL,
     nCreditos INTEGER NOT NULL,
     tipo      CHAR(11) NOT NULL
               CHECK(tipo IN(
               'Troncal','Obligatoria', 'Optativa'))
);

CREATE TABLE alumnos (
     idAlumno       INTEGER PRIMARY KEY,
     nombre         CHAR(40) NOT NULL,
     direccion CHAR(80),
     tfno      NUMERIC(9)
);

CREATE TABLE cursos (
     idCurso        INTEGER PRIMARY KEY
);
```

Suponemos un atributo nota que es propio de la interrelación matrícula y que añadimos a la tabla:

```
CREATE TABLE matrícula (
     idAsignatura   INTEGER REFERENCES asignaturas
                    ON DELETE CASCADE ON UPDATE CASCADE,
     idAlumno       INTEGER REFERENCES alumnos
                    ON DELETE CASCADE ON UPDATE CASCADE,
     idCurso        INTEGER REFERENCES cursos
                    ON DELETE CASCADE ON UPDATE CASCADE,
     nota      NUMERIC( 4, 2),
     PRIMARY KEY ( idAsignatura, idAlumno, idCurso)
);
```

En esta solución, no se puede dar de alta una matrícula sin que antes este dado de alta el curso. Pero esto puede que nos interese para que nadie meta por equivocación una matrícula en un curso inexistente.
Sin embargo, es posible que por el contrario, **no** nos haga falta controlar si el curso es un curso que se haya introducido previamente. Esto nos lleva a una tabla *matrícula* como la siguiente:

```
CREATE TABLE matrícula (
     idAsignatura   INTEGER REFERENCES asignaturas
                    ON DELETE CASCADE ON UPDATE CASCADE,
     idAlumno       INTEGER REFERENCES alumnos
                    ON DELETE CASCADE ON UPDATE CASCADE,
     idCurso        INTEGER ~~REFERENCES cursos~~
                    ~~ON DELETE CASCADE ON UPDATE CASCADE~~,
     nota      NUMERIC( 4, 2),
     PRIMARY KEY ( idAsignatura, idAlumno, idCurso)
);
```

Con lo cual carece de sentido tener la tabla de *cursos*, pues la información que contiene, ya estaría en *matrícula*, basta con hacer:

```
SELECT DISTINCT idCurso FROM matrícula;
```

para obtenerla. Además, la tabla *cursos* ya tampoco sirve para representar ninguna restricción que queramos mantener. La clave ajena de *matrícula REFERENCES cursos* hemos decidido quitarla, y la clave primaria de *cursos* ya no tiene utilidad.

Por tanto, en ese caso (el cliente piensa que no es necesario meter el curso antes que la matrícula), podemos simplificar quitando la tabla de *cursos*, y la clave ajena que liga *matrícula* con *cursos.*

Esta semántica, por la que cursos se subsume en matrícula, no tiene representación mediante cardinalidades de Chen, pero existen otras cardinalidades (cardinalidades de Merise [McAllister 1995]) que podrían recogerla. No obstante, en este curso no entraremos en las cardinalidades de Merise.

¿Cómo saber cuando me conviene más una solución que otra?, ¿mantener la tabla que representa el tiempo (e.g., *cursos*) o quitarla?. Tener una tabla específica para representar la dimensión temporal es adecuado en el caso en el que los acontecimientos temporales sucedan muy de cuando en cuando, son unos pocos, etc.. como los cursos.

Otra situación similar es cuando ese acontecimiento temporal se quiere reservar, como cuando se reserva una cita en una agenda. Si los huecos posibles de la agenda son filas de una tabla contra la que existe una clave ajena, obligamos a que el hueco esté dado de alta primero, evitando que los acontecimientos sucedan en instantes no permitidos por la agenda. Por ejemplo, en un torneo de tenis las fechas están prefijadas:

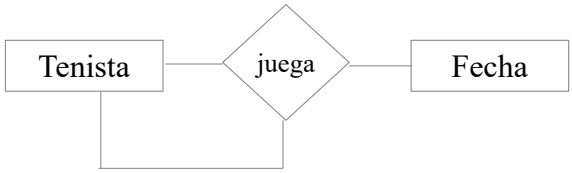


Figura 72: La dimensión temporal de los partidos de un torneo requiere tabla para evitar que se jueguen en fechas fuera del torneo.

Una tabla de fechas evitaría que un partido fuera programado fuera de las fechas permitidas.

Sin embargo, hay situaciones que piden a gritos la aproximación contraria, son aquellas en las que la dimensión temporal es cotidiana e imprevisible, y por tanto bien no merece la pena, o bien es imposible registrar y/o saber de antemano en que fecha van a suceder los acontecimientos. Por ejemplo:

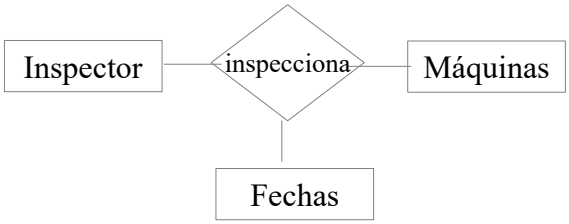


Figura 73: Caso en el que mantener una tabla de fechas no tiene sentido.

En este caso puede haber miles de fechas en las que se realicen inspecciones, pueden ser en principio cualquier fecha del año, ¿qué interés tiene meter todas las fechas del año, a su vez correspondientes varios años en una tabla con ese único campo?, ¿qué se supone que controlaríamos así?. Y sin embargo complicaríamos la operativa, pues cada vez que insertáramos una inspección, tendríamos que comprobar si esa fecha está dada de alta en la tabla de fechas, y si no hacer otra inserción en esa tabla insertándola. Parece entonces, más adecuado hacer la simplificación que nos liberaría de tener que mantener la tabla de *fechas.*

8. Agregaciones

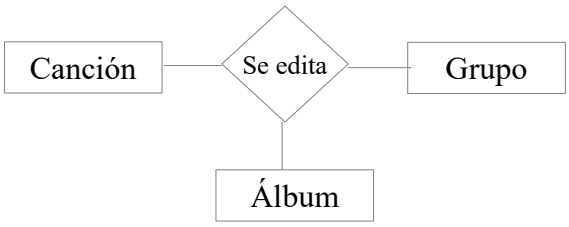
En una interrelación ternaria *R* entre las entidades *A, B* y *C* cada ocurrencia de *R* siempre relaciona un valor de cada uno de los tres elementos. Cabría preguntarse si el diagrama E/R nos provee de algún mecanismo mediante en cual, algunas ocurrencias de *R*, les pudiera faltar el valor quizás de *C*. Para este tipo de cosas utilizaremos las llamadas *agregaciones*.

Una agregación se forma por dos o más entidades interconectadas entre sí por una interrelación. Las dos o más entidades, junto la interrelación que las conecta se encierran en un rectángulo al que se da un nombre; y que denota que se trata de una agregación.

La agregación (ese rectángulo que encierra dentro una interrelación) se comporta entonces como una entidad mas del diagrama, por lo que puede tener atributos propios, y a su vez relacionarse con otras entidades del diagrama. Por tanto, **una agregación es una conversión de una interrelación en entidad**. Por ejemplo:

Supongamos que queremos hacer una base de datos musical. Hay una entidad *Canciones*, otra *Grupo Musical* y una tercera *Álbum*. Suponemos una relación binaria N:M entre *canciones* y *grupos* (i.e., no se puede repetir el par <*canción_x*, *grupo_y*>, es decir, para hacerlo más sencillo no consideramos el caso de que un mismo grupo haga varias versiones de la misma canción). Cada interpretación de una canción por un grupo puede editarse en varios álbumes, y cada álbum puede contener varias de estas interpretaciones.

Una posibilidad hubiera sido verlo como una ternaria:



Donde consideramos todas las cardinalidades máximas de Chen N. Es muy raro que un álbum contenga varias veces la misma canción interpretada por varios grupos, pero aún así no hemos descartado esa posibilidad, que es la que causaría una cardinalidad máxima de Chen N en el lado *Grupos.* Las otras 2 cardinalidades N son obvias.

Sin embargo, consideremos que un grupo podría interpretar una canción (quizás en sus directos, o quizás la tiene como maqueta) pero todavía no está editada en ningún álbum. Si fuese una ternaria, esas canciones no pueden relacionarse con grupo hasta que no hayan sido editadas en un álbum. Date cuenta que la tabla *se_edita* tendrá una

clave primaria compuesta por los AIPs de *canción, álbum* y *grupo,* lo que exige que en la tabla resultante las tres referencias a esos AIAs sean no nulas, por lo que no podemos insertar una fila en *se_edita* con el álbum vacío/nulo para representar que el grupo todavía no la ha editado.

Lo que realmente ocurre es que hay dos interrelaciones: (i) una interrelación *interpreta* entre canciones y grupos, y otra *edita en* entre las interpretaciones y los álbumes. Por ejemplo, como en la siguiente figura.

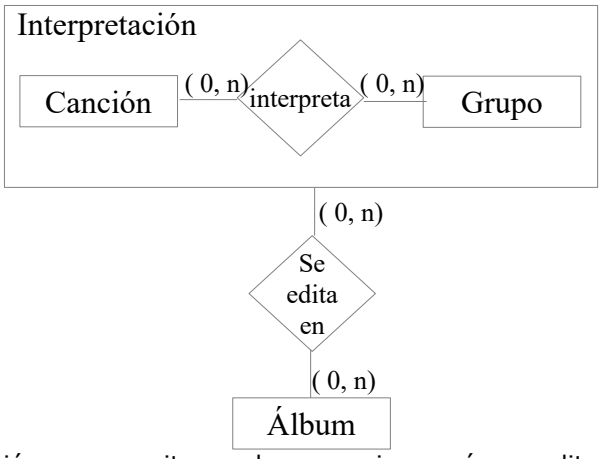


Figura 74: Agregación que permite que haya canciones aún no editadas en álbumes.

Hemos convertido la interrelación entre canciones y grupos en una agregación (i.e., en una entidad) llamada *interpretación*, la cual mantiene otra relación M:N con los grupos, dado que una interpretación puede aparecer en varios álbumes, y un álbum contiene varias interpretaciones.

Para convertir la interrelación en entidad se dibuja un rectángulo que englobe la interrelación y las entidades que relaciona. A veces, eso da lugar a dibujos complicados, y podemos encontrar textos [Costal-Costa 2005] que para evitar un exceso de líneas cruzándose, prefieren rodear tan solo el rombo como ilustra la Figura 75. En ese caso debemos dejar claro qué flechas se quedan fuera del rombo, y cuáles penetran en él.

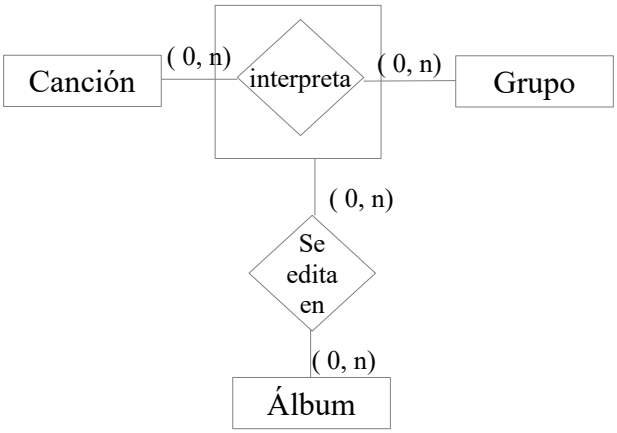


Figura 75: Agregación que permite que haya canciones aún no editadas en álbumes dibujada de otra forma.

Ahora una maqueta quedaría reflejada en la binaria *interpreta* pero no participaría en *se_edita_en*.

La agregación (*Interpretación*) **y la interrelación que encuadra** (*interpreta*) **son lo mismo**. Por cada elemento del conjunto de entidades *Interpretación* existe un solo elemento/flecha en la interrelación *Interpreta*, y viceversa. **Por ello, cuando pasemos ese diagrama a tablas NO TENDREMOS UNA TABLA *INTERPRETA* Y OTRA TABLA *INTERPRETACIÓN*, SINO UNA ÚNICA TABLA** (da igual el nombre).

De hecho da lo mismo colgar un atributo de la agregación *Interpreta*, que de la interrelación *Interpretación.* Por ejemplo, la duración de la interpretación (una misma canción puede durar distinto interpretada por distintos grupos),

podría representarse indistintamente de estas dos formas:

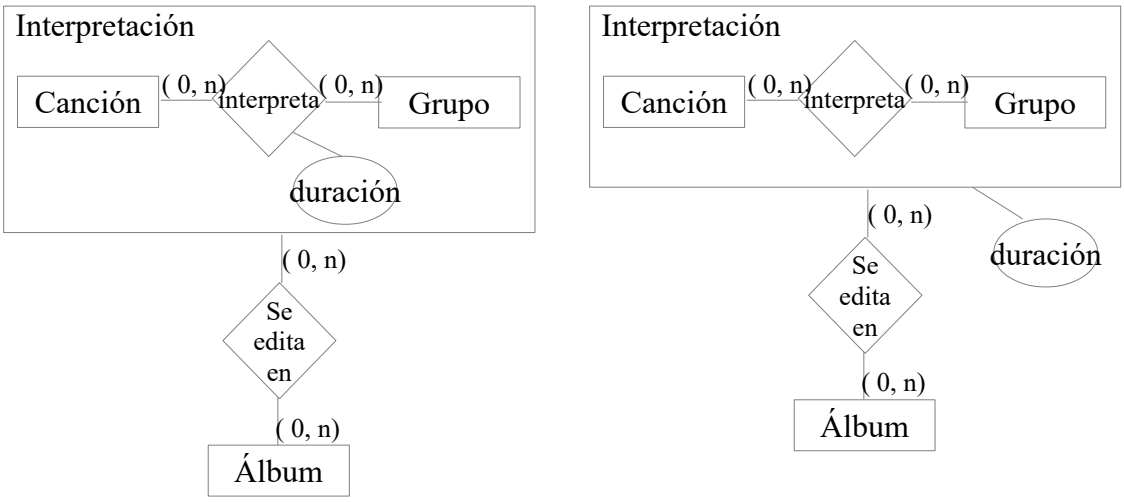


Figura 76: Como la agregación y la interrelación son los mismo, da igual de donde colgar el atributo *duración.*

Por todo ello, para pasar esquemas con agregaciones a tablas haremos lo siguiente:

Paso 1. Cada entidad (no-agregación) genera su propia tabla aplicando todas las reglas vistas hasta ahora.

Paso 2. Las interrelaciones M:N, ternarias, y *n*-arias entre entidades generan sus propias tablas con las reglas vistas hasta ahora.

La agregación no genera ninguna tabla adicional, pues se corresponde con la interrelación que engloba (en el ejemplo *interpreta*), y esa interrelación se supone que ya habría generado una tabla en el Paso 2.

Paso 3. Si la agregación tuviese atributos se añadirían a la tabla de la interrelación que engloba (e.g., la *duración* se añadiría a la tabla *interpreta* en el caso de la parte derecha de la Figura 76).

Paso 4. Si la agregación participase a su vez en otras interrelaciones, esas interrelaciones se tratarían como otras interrelaciones cualquiera, sólo que uno de los extremos es una entidad es especial, (i.e., es la agregación), y por ello tengo que tener en cuenta que su clave primaria es la que resulta de la interrelación que lleva dentro.

En nuestro ejemplo, la clave primaria de *Interpretación* (o *interpreta)* es la compuesta por *idGrupo* e *idCanción*, por lo que la tabla que genera interrelación M:N *se_edita* tendrá una clave ajena hacia *Interpretación* (o *interpreta)* compuesta por esos dos atributos; y su clave primaria tendrá esos dos atributos más la clave de *Álbum.*

De todo ello se deducen para el ejemplo las tablas *canción*, *grupo* y *álbum* de manera trivial (Paso 1 de la lista), y las tablas *Interpretación* (Paso 2 de la lista), con el atributo *duración* (Paso 3 de la lista) y la tabla *se_edita* (Paso 4 de la lista). Estas dos últimas tablas se crearían en SQL como sigue:

```
CREATE TABLE interpretaciones (
     idCancion NUMERIC(3) REFERENCES canciones,
     idGrupo        NUMERIC(3) REFERENCES grupos,
     duración_sg    INTEGER,
     PRIMARY KEY ( idCancion, idGrupo )
);
```

```
/*Duración (en sg=segundos) bien es atributo de la
agregación, bien del rombo interpreta (da igual).*/
CREATE TABLE se_edita (
     idCancion NUMERIC(3),
     idGrupo        NUMERIC(3),
     idAlbum        NUMERIC(2) REFERENCES álbumes
          ON DELETE CASCADE ON UPDATE CASCADE,
     FOREIGN KEY ( idCancion, idGrupo)
          REFERENCES interpretaciones
          ON DELETE CASCADE ON UPDATE CASCADE,
     PRIMARY KEY ( idCancion, idGrupo, idAlbum )
);
```

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

***Ejercicio**: Haz un diagrama E/R genérico correspondiente a una interrelación agregación que mantiene una relación R1 de cardinalidad 1:N con una entidad E1, y otra interrelación R2 de cardinalidad N:M con otra entidad E2. Utiliza letras para designar los nombres de la sentidades, relaciones y atributos que necesites. Deduce las dependencias funcionales y las claves. ¿A que forma normal llegas y por qué?.*

9. Las interrelaciones ISA

Las interrelaciones de ***especialización-generalización*** son una extensión del modelo E/R, que permiten expresar de una manera específica un tipo de relaciones que se dan mucho en la realidad, que son aquellas entre un conjunto genérico y otro que es una especialización del mismo.

Por ejemplo, supongamos el conjunto de los animales y de los mamíferos. Podemos decir:

1. Que un mamífero es una especialización (un caso específico de animal).
2. Que un animal es una generalización de los mamíferos.
3. Que todo mamífero es animal, y que por lo tanto la especialización es un subconjunto de la generalización.

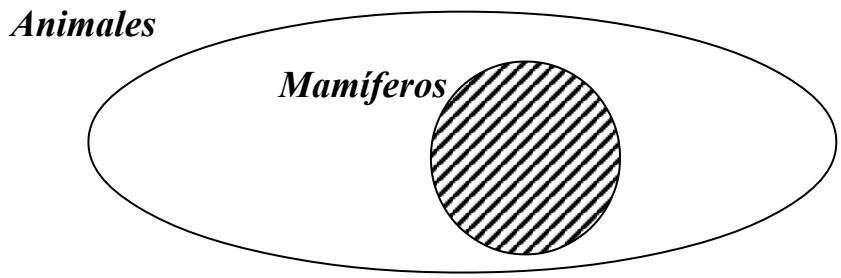


Figura 77: Los elementos de una especialización están incluidos dentro de la generalización.

4. O dicho de otra manera: que todo mamífero es a la vez un animal, y por tanto si desdoblásemos mamíferos y animales en dos conjuntos, estarían relacionados a través de una interrelación 1 a 1. A un mamífero le corresponden (1, 1) animales, pero a un animal le corresponden (0, 1) mamíferos. El cero viene porque,

por ejemplo un cocodrilo siendo animal no se relaciona con ningún mamífero (no es un mamífero).

5. Que todas las propiedades de un animal también las tiene (las ***heredan***) los mamíferos (los animales se reproducen, se alimentan etc...). Luego todas las propiedades de la generalización son heredadas por la especialización.

Por tanto, podríamos haber representado la interrelación entre mamíferos y animales como una interrelación 1:1. El mayor inconveniente hubiera sido que todos los atributos de los animales hubieran tenido que volver a pintarse en los mamíferos. Por ello, es más recomendable pintarlo como un tipo especial de interrelación que llamaremos ***ISA*** (del inglés: "*is a*", en español: "*es un*") o interrelación de generalización-especialización.

> **Nota**: También se dice que la especialización es un ***subtipo*** de la generalización, y que la generalización es un ***supertipo*** de la especialización. Esta nomenclatura es más habitual cuando se habla de tipos de entidades, en lugar de conjuntos de entidades como hemos hecho nosotros.

9.1.1 Representación diagramática de las relaciones ISA

Hay fundamentalmente dos estilos. En ambos casos la interrelación, en lugar de representarse por un rombo, se representa a través de un triángulo a modo de flecha. En el estilo más tradicional, esta flecha apunta a la especialización. Sin embargo, por influencia de otro tipo de diagramas (UML) cada vez hay más gente que lo hace al revés (apuntando a la generalización) [Connolly y Begg 2005], [Ullman y Widom 1997]. Nosotros adoptaremos el estilo tradicional que también es el usado en MÉTRICA [METRICA V3 2001], con lo que el ejemplo anterior quedaría como muestra la figura:

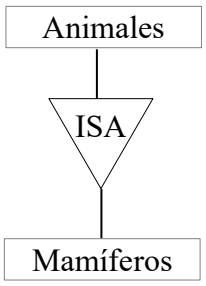


Figura 78: Representación diagramática de las relaciones ISA.

El triángulo de la interrelación ISA apunta a los mamíferos que son la especialización. Como ves **no se especifica la cardinalidad, pues está implícita**, siempre va a ser la misma cuando hay una interrelación ISA. Es como si pusiéramos:

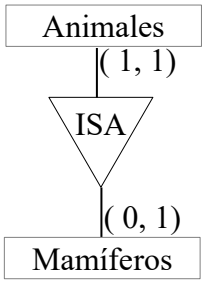


Figura 79: Las relaciones ISA llevan implícita siempre **la cardinalidad** Generalización (1,1) - Especialización (0,1), y por eso **nunca se pone**. **(i.e., la figura muestra lo que no hay que hacer)**.

Pues como todo mamífero es animal, le corresponde siempre un animal, pero como no todo animal es mamífero, a veces le corresponderá un mamífero, pero no siempre.

9.1.2 *Herencia de Atributos e Interrelaciones*

Cambiemos de ejemplo, para ver qué pasa con los atributos. Supongamos que tenemos la generalización *Personas*, y la especialización *Casados*:

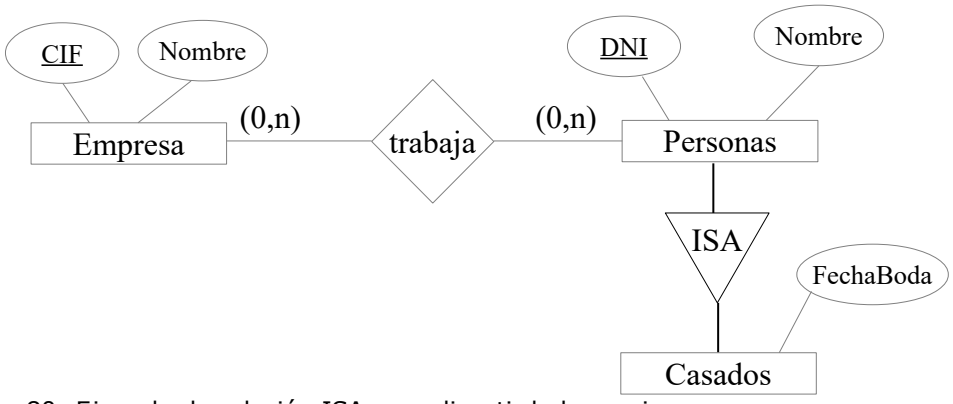


Figura 80: Ejemplo de relación ISA para discutir la herencia.

Las personas trabajan en empresas. Observa que:

1. La especialización *casados* tendría que tener el *DNI* y el *nombre* de la persona, pero no hay que ponerlo, porque ya lo hereda de su generalización, **sólo hay que pintarlo en la generalización**. Se dice que la especialización hereda los atributos de la generalización.
2. De la misma forma, podría parecer que *casados* es una entidad débil, sin AIP. No es así, si que lo tiene, lo hereda de *personas*, y por tanto su AIP es el *DNI.* Se dice que la especialización hereda los AIPs de la generalización.
3. Las *empresas* deberían estar relacionadas con las personas casadas también, y aparentemente no es así. Sin embargo, como todo casado "IS A" persona, los *casados* (especialización) heredan las relaciones de la generalización. Es decir, la relación *trabaja* también la heredan los *casados*, sin necesidad de pintarlo explícitamente.

9.2. Tipos de Interrelaciones de Especialización

Las interrelaciones de especialización se pueden clasificar de dos maneras:

9.2.1 *Con/sin solapamiento*

Si varias entidades son especializaciones de la misma generalización, puede ocurrir que existan elementos de la generalización que pertenezcan simultáneamente a varias de las especializaciones. En ese caso se dice que hay solapamiento. Por ejemplo: supongamos:

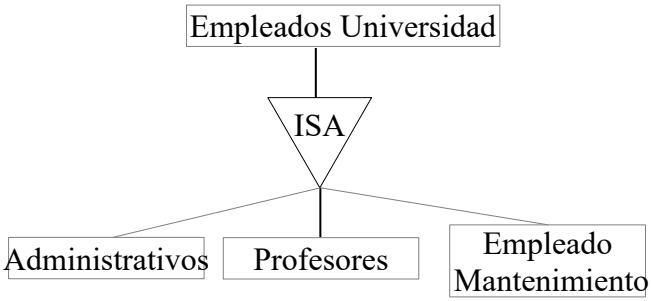


Figura 81: Ejemplo de interrelación ISA para discutir el solapamiento.

Si, por ejemplo, pudiese haber administrativos que a la vez fuesen profesores; o empleados de mantenimiento que a la vez fuesen administrativos etc... habría solapamiento. Si por el contrario, un administrativo no puede ser a la vez ni

profesor ni empleado de mantenimiento, y un profesor no puede ser tampoco empleado de mantenimiento, no hay solapamiento. Cuando hay solapamiento podría haber instancias que asumiesen todas las especializaciones simultáneamente; por ejemplo, podría haber un profesor que fuese también administrativo y empleado de mantenimiento.

En la figura anterior lo que se ha representado es el caso **CON** solapamiento. Cuando no hay solapamiento, se dice que la relación ISA es ***exclusiva***, y se representa con un arco, como en la figura:

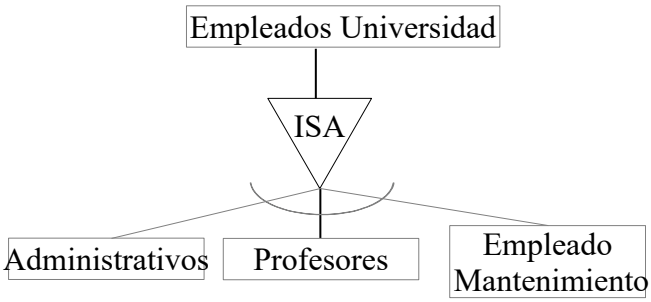


Figura 82: El "no-solapamiento" o exclusividad se representa mediante un arco.

Nota: El arco algunas veces no abarca todas las especializaciones. Si por ejemplo el arco sólo recorriese a los *Administrativos* y a los *Profesores*, significaría que es incompatible ser administrativo y profesor a la vez, pero que no es incompatible ser administrativo y empleado de mantenimiento a vez, ni ser profesor y empleado de mantenimiento a la vez.

9.2.2 Interrelaciones Totales vs. Parciales

Se dice que una interrelación es total, si la unión de los elementos de todas sus especializaciones tiene los mismos elementos que la generalización, en caso contrario se dice que es parcial.

En el ejemplo anterior, podemos preguntarnos si hay empleados de la universidad que no son ni administrativos, ni profesores, ni empleados de mantenimiento. En caso afirmativo, la relación es parcial, porque la unión de las tres especializaciones tendría menos elementos que la generalización. Si por el contrario, no pudiese haber empleados de la universidad que no estuviesen en uno de los tres casos, la interrelación sería total.

Tal y como estamos representando las interrelaciones hasta ahora, todas serían parciales. Si se quiere hacer notar que son totales, hay que poner una doble linea en el lado de la generalización [METRICA V3 2001], como en la figura:

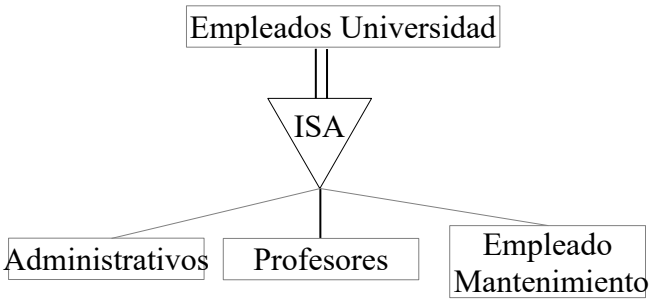


Figura 83: Las ISAs totales se representan con una doble línea encima del triángulo.

En ese caso estamos representando que es total con solapamiento. Si queremos total sin solapamiento sería añadir el arco:

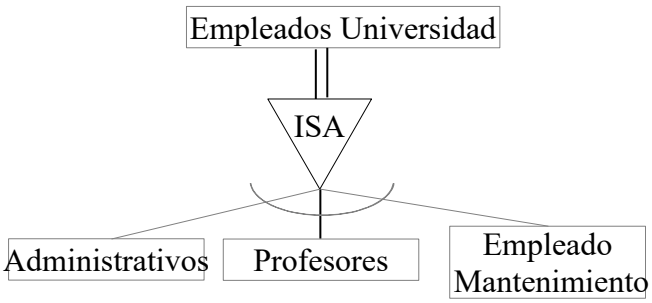


Figura 84: Ejemplo de empleados de la universidad como total sin solapamiento (o total exclusivo).

Por lo tanto, tenemos cuatro casos dependiendo si hay o no solapamiento, y de si es o no total.

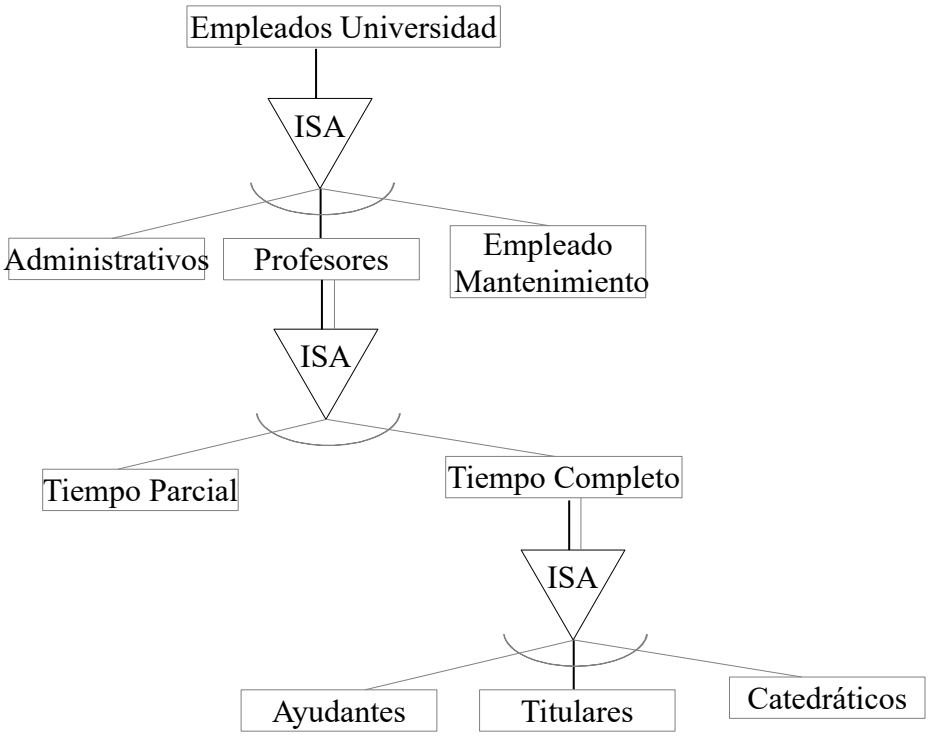


Figura 85: Ejemplo de jerarquía de herencia con 3 niveles.

9.2.3 Jerarquías de Herencia

Una especialización puede ser a su vez generalización de nuevas especializaciones, formando una jerarquía con más de un nivel de especialización. Por ejemplo (Figura 85), supongamos que los profesores, pueden tener dedicación a tiempo completo o parcial, y los a tiempo completo puedan ser a su vez: titulares, ayudantes o catedráticos, entonces tendríamos una jerarquía con tres niveles de especialización.

Hemos notado que casualmente cada nivel es sin solapamiento, el primer nivel (i.e., *Empleados - Universidad*) es el único que es parcial (por ejemplo podría haber entrenadores deportivos, asesores legales etc... que no corresponden a ninguna de las tres especializaciones de primer nivel).

9.2.3.1 Jerarquías paralelas y herencia múltiple (+)

Un conjunto de entidades puede ser generalización de varias jerarquías. Por ejemplo, que de manera paralela a la clasificación de los profesores como de tiempo completo o parcial, tenemos una clasificación respecto al cargo (e.g..

decano, subdirector, vicerrector etc...) vs. sin cargo, tal que así:

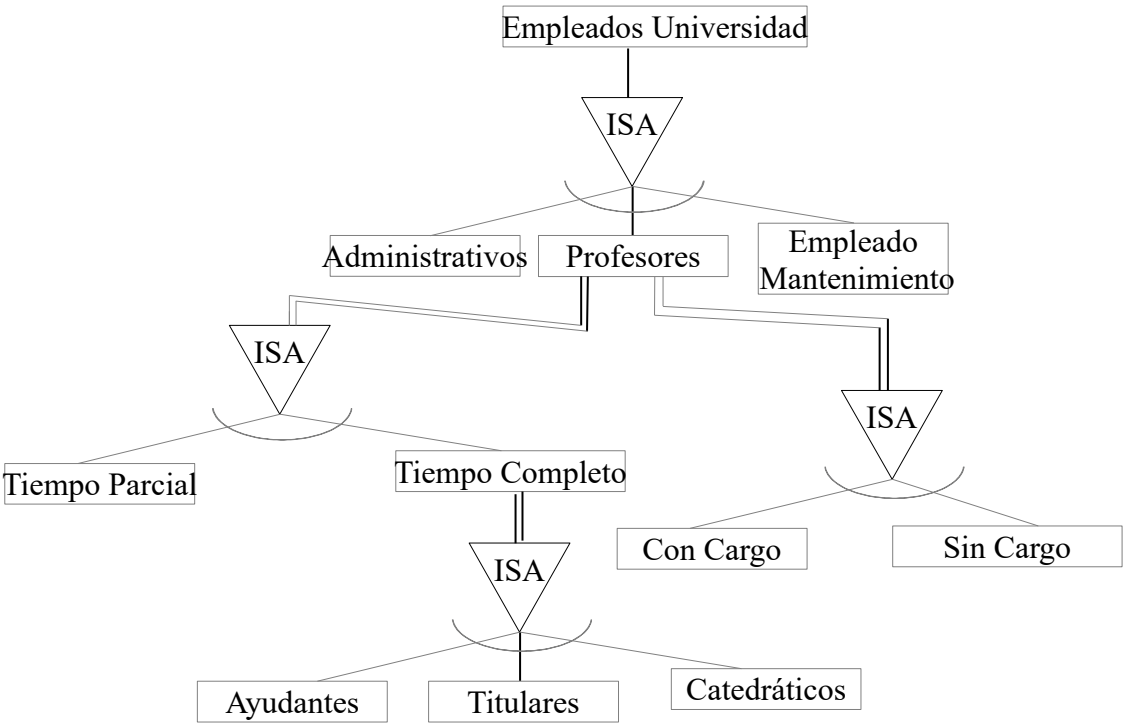


Figura 86: Ejemplo en el que en el nivel 2 aparecen dos jerarquías que son entre si paralelas.

De manera que podría haber profesores a tiempo parcial con o sin cargo, y todas las variantes a tiempo completo, con o sin cargo.

Un atributo de empleado de la universidad (e.g., el nombre del empleado) sería común a todas las especializaciones por herencia. Si un profesor tiene como atributo *años de experiencia docente* los titulares, ayudantes, catedráticos y tiempos parciales lo tendrán, tanto si tienen cargo como si no.

¿Podría una especialización, ser especialización de varias generalizaciones simultáneamente?. Es posible. Esto se conoce como ***herencia múltiple***, pues la especialización heredaría atributos de múltiples generalizaciones. En la práctica veremos pocos o ningún caso que requiera necesariamente de herencia múltiple y tiene gran cantidad de detractores.

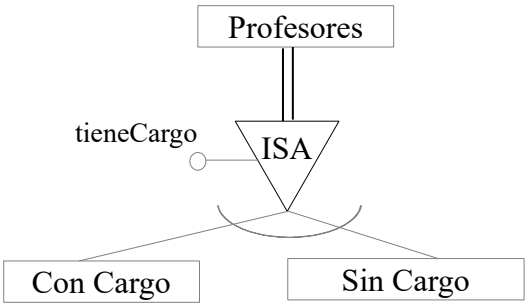


Figura 87: El atributo discriminante se representa colgando del triángulo.

9.3. Discriminante de una interrelación ISA

La división en distintas especializaciones puede venir dada a través del valor de un determinado atributo que llamaremos discriminante. Por ejemplo, la división entre "con cargo" y "sin cargo" podría venir dada por un atributo booleano *tiene_cargo* que podría valer verdadero o falso. El discriminante, se hace notar como atributo del triángulo

como en la Figura 87.

Los discriminantes booleanos son típicos de especializaciones dicotómicas, pero puede haber otro tipo de discriminantes. Por ejemplo, un atributo *tipo de actividad* podría haber sido el discriminante del primer nivel de la jerarquía (administrativos, profesores y empleados de mantenimiento), y ese tipo de actividad podría haber sido un texto con los valores *ADM, PROF* y *MANTEN* respectivamente, o quizás los números del 1 al 3, asignando a cada uno la semántica del tipo de empleado correspondiente.

El discriminante podría aparecer en casos con solapamiento. Por ejemplo, supongamos que permitiésemos que un profesor pudiese ser administrativo, y/o empleado de mantenimiento. Entonces *tipo_de_actividad* podría tomar los valores de todas las combinaciones posibles: *ADM, PROF, MANTEN*, pero además: *ADM-PROF, ADM-MANTEN, PROF-MANTEN* y *ADM-PROF-MANTEN*, o bien en este caso, por ejemplo, los números del 1 al 7.

No es obligatorio que las relaciones ISA tengan discriminante, pero hay casos en los que facilita la transformación en tablas relacionales, pues se pueden utilizar para controlar tanto el solapamiento y como si es total/parcial, como veremos más adelante.

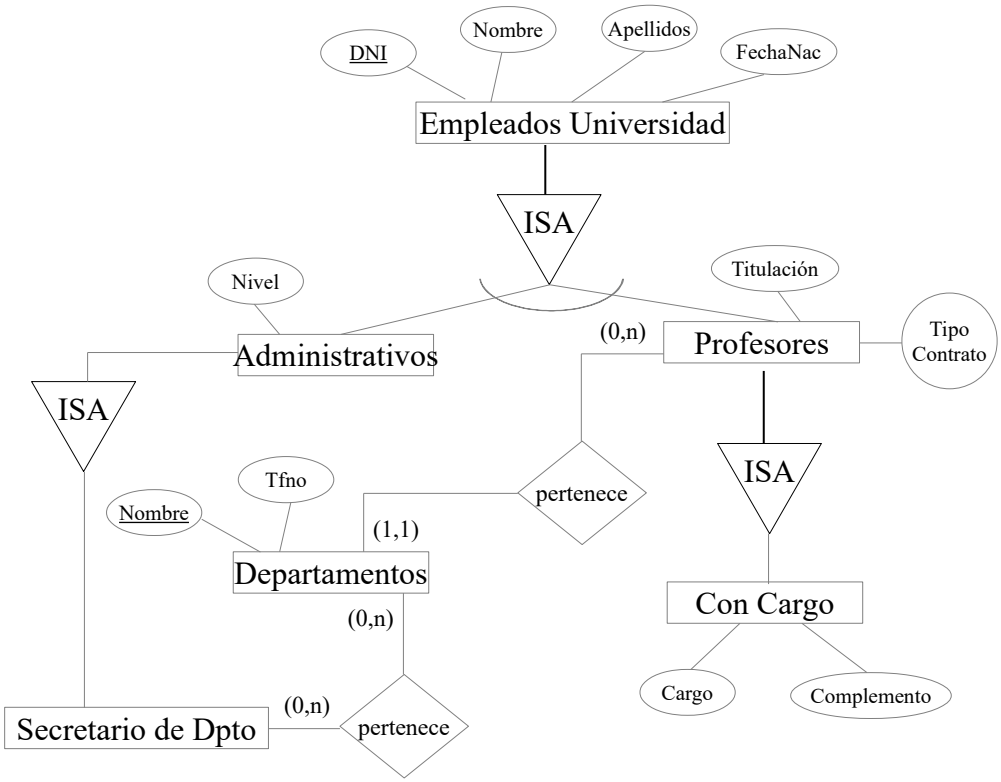


Figura 88: Jerarquía ISA de Ejemplo.

9.4. Paso a Tablas Relacionales de las Interrelaciones ISA

Existen tres formas de trasladar una interrelación ISA del modelo E/R a un conjunto de tablas relacionales. Cada una de esas fórmulas será más adecuada según qué caso.

9.4.1 Solución 1: Tratar las ISAs como una interrelación 1:1

Esta solución consiste en pensar que las ISAs son relaciones 1:1, y aplicar las reglas de transformación al uso. Es decir:

1. Creamos una tabla para la generalización, y otra tabla para cada una de las especializaciones.
2. La tabla de la generalización tendrá los atributos propios de la entidad generalización.

3. Cada tabla especialización mantendrá los atributos propios de la correspondiente entidad especialización, más la clave de la generalización que:
    1. Ejercerá de clave primaria en la especialización.
    2. Ejercerá también de clave ajena en la especialización. Lo usual es que se declare con borrado y actualización en cascada.

Utilicemos esta técnica con el ejemplo de la Figura 88[15]. La generalización tiene tabla propia con sus atributos:

```
CREATE TABLE empleados(
     DNI       NUMERIC(8) PRIMARY KEY,
     Nombre    CHAR(30) NOT NULL,
     Apellidos CHAR(30) NOT NULL,
     FechNac   DATE NOT NULL
);
```

(Asumamos que el nombre y apellidos son campos obligatorios)

Los profesores generan su propia tabla que se relacionaría como parte a varios con *departamentos*, por lo que declaramos primero *departamentos*:

```
CREATE TABLE departamentos(
     Nombre    CHAR(30) PRIMARY KEY,
     Tfono     NUMERIC(9)
);
```

La especialización *profesores*, es una tabla, con clave primaria *DNI* (ya que es la clave de su generalización). Con el DNI se relacionará con *empleados*, y con el nombre del departamento, con *departamentos.* En la relación con *departamentos* apreciamos una cardinalidad mínima uno, que podría servir de justificación para hacer borrado/actualización en cascada, nosotros no lo hemos considerado. Lo que si que exige esa cardinalidad mínima uno es hacer que el nombre del departamento sea NOT NULL.

```
CREATE TABLE profesores(
     DNI  NUMERIC(8) PRIMARY KEY
          REFERENCES empleados
          ON DELETE CASCADE ON UPDATE CASCADE,
     NombreDept CHAR(30) REFERENCES departamentos
                         NOT NULL,
     titulación     CHAR(30),
     tipoContrato   CHAR(10)
);
```

15 Algunas cosas de este ejemplo no son reales, como es el caso de que un secretario pueda ser compartido por varios departamentos. Nos hemos tomado esa licencia y otras a fin de tener un ejemplo lo más completo posible.

Como ya está declarada *profesores*, podríamos declarar la especialización *con cargo*:

```
CREATE TABLE cargos(
     DNI  NUMERIC(8) PRIMARY KEY
          REFERENCES profesores
          ON DELETE CASCADE ON UPDATE CASCADE,
     cargo     CHAR(30) NOT NULL,
     complemento    NUMERIC(5,2)
);
```

Ahora vamos con los administrativos:

```
CREATE TABLE administrativos(
     DNI  NUMERIC(8) PRIMARY KEY
          REFERENCES empleados
          ON DELETE CASCADE ON UPDATE CASCADE,
     nivel     CHAR(30)
);
```

Ahora la especialización, que no tiene atributos propios:

```
CREATE TABLE secretarios(
     DNI  NUMERIC(8) PRIMARY KEY
               REFERENCES administrativos
               ON DELETE CASCADE ON UPDATE CASCADE
);
```

Finalmente, la relación varios a varios, donde la referencia a secretarios viene dado por el DNI, que es realmente una clave que le viene heredada desde *empleados.*

```
CREATE TABLE secre_depar(
     DNI       NUMERIC(8) REFERENCES secretarios
               ON DELETE CASCADE ON UPDATE CASCADE,
     NombreDept CHAR(30) REFERENCES departamentos
               ON DELETE CASCADE ON UPDATE CASCADE,
     PRIMARY KEY (DNI, NombreDept)
);
```

La tabla *secretarios* parece que no aporta ningún dato que no tengamos ya en las otras tablas, sin embargo, tiene un papel importante en cuanto a que impide que un administrativo “no-secretario” sea asignado en la interrelación con

departamentos.

Como vemos si quisiéramos obtener un listado con todos los atributos de un administrativo, tendríamos que hacer un *join* entre *empleados* y *administrativos.*

Si hubiese discriminante, bajo esta solución podría omitirse su paso a relacional, pues el mero hecho de ser fila de una(s) especialización(es) ya supone un valor en el discriminante. Es decir, ser fila de *Administrativos* es otra forma de representar que el discriminante vale por ejemplo *ADM.* En el caso con solapamiento, ser fila de *Administrativos* y ser también fila de *Profesores*, es otra forma de representar que el discriminante vale por ejemplo *ADM-PROF.*

Bajo esta solución, nota que es imposible controlar que se violen las restricciones de exclusividad o de totalidad. Por ejemplo, aunque hayamos pintado que es incompatible ser profesor y administrativo a la vez, estas tablas no impiden que insertemos un profesor con el mismo DNI que un administrativo, violando así la exclusividad entre ambas especializaciones; y si *Empleados-Universidad* hubiera sido total, tampoco nada hubiese impedido que insertáramos un empleado cuyo DNI luego no apareciese en ninguna de las especializaciones, violando así la restricción de ser una generalización total.

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

9.4.2 Solución 2: Juntar especializaciones y generalización en una sola tabla

Esta solución consiste en:

1. Crear una única tabla en la que tendrá una fila por cada elemento de la generalización.
2. La tabla tendrá todos los campos de la generalización, más todos los campos de cada una de las especializaciones.
3. Es muy útil además añadir el discriminante para determinar a qué especialización (o especializaciones en el caso solapado) corresponde esa fila.

Con el mismo esquema ejemplo que hemos utilizado para la solución uno, se perfilarían tres tablas:

1. Una tabla que aglutinase toda la jerarquía ISA.
2. Una tabla de departamentos.
3. Una tabla con la relación N:M entre secretarios y departamentos.

Como los profesores referencian a *departamentos,* los departamentos tienen que crearse antes que la tabla de *empleados*

```
CREATE TABLE departamentos(
     Nombre    CHAR(30) PRIMARY KEY,
     Tfono     NUMERIC(9)
);
```

Ahora creamos una tabla con todos los campos de la jerarquía de herencia de *empleados.* Es muy útil añadir el discriminante, pues si vale '*A*' ya nos dice que es un administrativo no-secretario, si vale '*S*' es un secretario, si vale '*P*' un profesor sin cargo,y si vale '*C*' con cargo. Hemos añadido una quinta opción '*E*' que representa al empleado que no pertenezca a ninguna de las otras cuatro especializaciones, ya que en nuestro caso la ISA es parcial, si fuese total, no hubiera sido necesario añadir el caso '*E*'.

El *CHECK* final podría simplificarse aplicando leyes de la lógica booleana, pero hemos preferido dejarlo así por claridad. Nota que usamos siempre IS NULL, recuerda que "=NULL" sólo puede devolver NULL, y nunca puede devolver *verdadero* o *falso*. Este *CHECK* gigante según el valor del discriminante hace que:

1. Los campos correspondientes a otras especializaciones distintas a la que marca el discriminante para cada

fila sean nulos. Por ejemplo, si el discriminante vale '*P*', se trata de un profesor sin cargo, y por lo tanto <*cargo* IS NULL>, pues el atributo cargo no es de la especialización '*P*'. Lo mismo ocurre con el resto de campos que no son propios de un '*P*' serán nulos (i.e., *complemento* y *nivel*). También ocurre lo mismo en el caso '*E*' donde han de hacerse nulos todos los atributos de las especializaciones.

2. Los campos de una especialización que sean obligatorios se puedan controlar como no nulos específicamente para esa especialización. Por ejemplo, si el discriminante vale '*C*', se trata de un profesor con cargo, y por lo tanto <*cargo* IS NOT NULL>, pues el atributo *cargo* existe para la especialización '*C*' y además es obligatorio en esa especialización.
3. Los campos de una especialización que no sean obligatorios se puedan controlar como no obligatorios específicamente para esa especialización. Por ejemplo, si el discriminante vale '*A*' o '*S*', se trata de un administrativo o secretario. Ambos tienen el campo *nivel.* Supongamos que el enunciado del problema nos dice que ese campo no es obligatorio (i.e., puede valer nulo). Entonces **ese campo no aparece en el *CHECK* para el término *(tipo= 'A' OR tipo= 'S')***, pues cuando fila sea de alguno de esos dos tipos, *nivel* puede tomar un valor tanto nulo, como no nulo, y por lo tanto no hay que chequear nada.

El discriminante le hemos hecho NOT NULL. Podrías quizás pensar en hacerle *nullable* y aprovechar el valor nulo, por ejemplo, para representar el caso '*E*'. Si lo haces así, sustituye el "*OR (tipo='E'*" por "*OR (tipo IS NULL*". No obstante, nosotros no seguiremos esta práctica para evitar problemas con los nulos.

```
CREATE TABLE empleados(
      DNI          NUMERIC(8) PRIMARY KEY,
      Nombre       CHAR(30) NOT NULL,
      Apellidos    CHAR(30) NOT NULL,
      FechNac      DATE NOT NULL,
      tipo         CHAR(1) NOT NULL
                   CHECK(tipo IN ('A','P','S','C','E')),
      NombreDept         CHAR(30) REFERENCES departamentos,
      titulación         CHAR(30),
      tipoContrato       CHAR(10),
      cargo              CHAR(30),
      complemento        NUMERIC(5,2),
      nivel              CHAR(30),
      CHECK ( (tipo='P' AND NombreDept IS NOT NULL
                  AND cargo IS NULL
                  AND complemento IS NULL
                  AND nivel IS NULL)
            OR
            (tipo='C' AND NombreDept IS NOT NULL
                   AND cargo IS NOT NULL
                   AND nivel IS NULL)
            OR ((tipo= 'A' OR tipo= 'S' )
                  AND NombreDept IS NULL
                  AND titulacion IS NULL
                  AND tipoContrato IS NULL
                  AND cargo IS NULL
                  AND complemento IS NULL)
            OR (tipo='E'
                  AND NombreDept IS NULL
                  AND titulacion IS NULL
                  AND tipoContrato IS NULL
                  AND cargo IS NULL
                  AND complemento IS NULL
                  AND nivel IS NULL)
      )
);
```

Para generar ese *CHECK* final, si no tenemos mucha práctica, puede ser útil hacernos primero una tabla como la de la Figura 89. Cada fila de la tabla representa un posible valor del disciminante, y cada columna cada posible atributo **de las especializaciones, incluyendo las claves ajenas**.

	NombreDept	cargo	complemento	nivel	titulación	tipo contrato
Profesor tipo='P'	NOT NULL	No tiene	No tiene	No tiene	NULLABLE	NULLABLE
Prof. con cargo tipo='C'	NOT NULL	NOT NULL	NULLABLE	No tiene	NULLABLE	NULLABLE
Administrativo tipo= 'A'	No tiene	No tiene	No tiene	NULLABLE	No tiene	No tiene
Secretario tipo= 'S'	No tiene	No tiene	No tiene	NULLABLE	No tiene	No tiene
Empleado genérico tipo='E'	No tiene	No tiene	No tiene	No tiene	No tiene	No tiene

Figura 89: Tabla para ver qué campos son *nullables* y no - *nullables* según lo que valga el discriminante.

En la tabla marcamos:

1. con NOT NULL aquellas celdas en el que el campo de la columna sería obligatorio para el valor del discriminante en la fila. Cuidado con los NOT NULL que generan las cardinalidades mínimas "1" en los extremos con cardinalidad (1,1) de las interrelaciones 1:1 y 1:N. Por ejemplo, el NOT NULL de *nombreDept* es porque se da esta circunstancia en la interrelación entre *profesores* y *departamentos* .
2. con *No tiene* aquellas celdas en el que el campo de la columna no debiera de existir para el valor del discriminante en la fila. (Nota que la generalización siempre toma el valor *no tiene* en todas las columnas, pues no tiene atributos de las especializaciones).
3. y con *nullable* el resto de las celdas, que son en las que el campo de la columna sí que existe para el valor del discriminante en la fila, pero no tenemos ninguna restricción que impida que valga nulo o no. Cuidado con los *nullables* que generan las cardinalidades mínimas "0" en los extremos con cardinalidad (0,1) de las interrelaciones 1:1 y 1:N. Por ejemplo, si la interrelación entre *profesores* y *departamentos* hubiese tenido un (0,1) en el lado de *departamentos,* los NOT NULL de la columna *nombreDept* hubiesen sido *nullables.*

Una vez tienes la tabla, el CHECK final es inmediato:

1. Cada casilla NOT NULL se sustituye por un término *AND <campo de la columna> IS NOT NULL* en el término OR correspondiente al valor del discriminante en la fila.
2. Cada casilla *No tiene* se sustituye por un término *AND <campo de la columna> IS NULL* en el término OR correspondiente al valor del discriminante en la fila.
3. Cada casilla *nullable* se ignora.

Varias advertencias **IMPORTANTEs**:

1. **No pongas nunca el NOT NULL en el REFERENCES de una interrelación de la especialización**. Por ejemplo, cuando hemos declarado el campo "*NombreDept CHAR(30) REFERENCES departamentos*" observa que no le hemos puesto como NOT NULL pese a que la interrelación entre *profesores* y *departamentos* tiene un extremo (1, 1). Esto es debido a que solamente son los profesores los que participan en esa interrelación, y eso ya lo hemos controlado con el *"NombreDept IS NOT NULL"* en los tipos *P* y *C*. Si pusiéramos el NOT NULL junto al REFERENCES obligaríamos a que todos los empleados tuvieran ese campo lleno, incluyendo a los que no son profesores. Si que habría que haber puesto el NOT

NULL junto al REFERENCES si la interrelación en lugar de haber sido de una especialización (e.g., *profesores*), hubiese sido de la generalización (e.g., *empleados*). Por ejemplo, si la generalización *empleados-universidad* participara en una interrelación 1:N con una entidad *provincias*, y a cada empleado le correspondiese obligatoriamente una provincia (lo caul marcaríamos con una cardinalidad (1,1) en el extremo de *provincias*), entonces el campo *provincias* de *empleados* si que tendría el NOT NULL junto al REFERENCES, porque ahora son todos los empleados, independientemente de su especialización, los que han de relacionarse obligatoriamente con una provincia.

2. **Si la especialización es total**, el único cambio que hay que hacer es quitar el caso de la generalización, *E* en nuestro ejemplo, tanto en el *CHECK IN* del campo discriminante, como en el *CHECK* final; pues en la tabla de la Figura 89 desaparecería la fila del caso E.
3. **Si hay especializaciones solapadas**, por ejemplo que se pueda ser administrativo y profesor a la vez, entonces en el CHECK del discriminante surgirían nuevos casos correspondientes a las nuevas especializaciones combinadas. En nuestro caso podríamos hacer:

```
tipo       CHAR(2) NOT NULL
                CHECK(tipo IN ('A','P','S','C','E',
                    'AP', 'AC', 'SP', 'SC')),
```

de manera que: AP = Administrativo no – secretario que es a la vez profesor sin cargo, AC = Administrativo no – secretario que es a la vez profesor con cargo, SP =Secretario que es a la vez profesor sin cargo y SC = Secretario que es a la vez profesor con cargo.

Esto generaría cuatro nuevas filas en la tabla de la Figura 89:

	NombreDept	cargo	complemento	nivel	titulación	tipo contrato
AP	NOT NULL	No tiene	No tiene	NULLABLE	NULLABLE	NULLABLE
AC	NOT NULL	NOT NULL	NULLABLE	NULLABLE	NULLABLE	NULLABLE
SP	NOT NULL	No tiene	No tiene	NULLABLE	NULLABLE	NULLABLE
SC	NOT NULL	NOT NULL	NULLABLE	NULLABLE	NULLABLE	NULLABLE

Figura 90: Nuevas filas que aparecerían en la tabla de la si la especialización entre profesores y administrativos fuese solapada.

Estas filas son fáciles de deducir, los NOT NULL y los NULLABLEs se propagan desde cada una de las especializaciones puras hasta las especializaciones mixtas, y en caso de conflicto se propaga el NOT NULL. El resto de celdas se rellenan con *no tiene*.

La tabla de la figura ampliaría el CHECK final de la siguiente forma:

```
OR (tipo = 'AP' OR tipo='SP'
      AND NombreDept IS NOT NULL
      AND cargo IS NULL AND complemento IS NULL)
OR (tipo = 'AC' OR tipo='SC'
     AND NombreDept IS NOT NULL
     AND cargo IS NOT NULL)
)
```

4. Otra observación interesante es que el *CHECK(tipo IN ('A','P','S','C','E'))* podría omitirse pues es redundante, ya que el CHECK del final no deja que *tipo* valga otra cosa. No obstante, se ha mantenido en la figura para dar mayor claridad al CREATE TABLE. Lo que no debes de quitar alegremente es el NOT NULL del campo *tipo.*

Finalmente, implementamos la tabla correspondiente la interrelación N:M entre departamentos y secretarios. En este ejemplo vemos como no es posible controlar que el DNI fuese de un empleado que no fuese secretario.

```
CREATE TABLE secre_depar(
     DNI  NUMERIC(8) REFERENCES empleados
               ON DELETE CASCADE ON UPDATE CASCADE,
     NombreDept CHAR(30) REFERENCES departamentos
               ON DELETE CASCADE ON UPDATE CASCADE,
     PRIMARY KEY (DNI, NombreDept)
);
```

Existen formas de resolver este problema pero son muy artificiosas, por lo que no lo llevaremos a la práctica.

***Ejercicio**: Comprueba en qué forma normal están las tablas obtenidas.*

9.4.3 Solución 3: Hacer una tabla independiente por cada especialización (+)

Esta solución consiste en:

1. Crear una tabla en cada especialización incluyendo todos los campos (tanto los propios como los heredados (claves ajenas incluidas)).
2. Y si la ISA es parcial, crear otra tabla para la generalización, en la que sólo irían filas correspondientes a elementos que no pertenecen a ninguna de las especializaciones (i.e., si la ISA es total, al no existir ninguno de estos elementos, no es necesario crear esta última tabla).

Para el ejemplo de siempre tenemos:

1. Tablas de la jerarquía ISA
    1. Por ser parcial, una tabla de empleados que no son ni profesores ni administrativos, no habría que crearla si esta ISA hubiera sido total.
    2. Otra de administrativos que no son secretarios más otra de secretarios
    3. Otra de profesores sin cargo más otra de profesores con cargo
2. La tabla de departamentos y la de la relación N:M.

```
CREATE TABLE empleados(
     DNI            NUMERIC(8) PRIMARY KEY,
     Nombre         CHAR(30) NOT NULL,
     Apellidos      CHAR(30) NOT NULL,
     FechNac        DATE NOT NULL
);
```

```
CREATE TABLE departamentos(
     Nombre    CHAR(30) PRIMARY KEY,
     Tfono     NUMERIC(9)
);

CREATE TABLE profesores(
     DNI            NUMERIC(8) PRIMARY KEY,
     Nombre         CHAR(30) NOT NULL,
     Apellidos CHAR(30) NOT NULL,
     FechNac        DATE NOT NULL,
     NombreDept     CHAR(30)
                     REFERENCES departamentos
                     NOT NULL,
     titulación     CHAR(30),
     tipoContrato   CHAR(10)
);

CREATE TABLE cargos(
     DNI                NUMERIC(8) PRIMARY KEY,
     Nombre             CHAR(30) NOT NULL,
     Apellidos CHAR(30)  NOT NULL,
     FechNac            DATE NOT NULL,
     NombreDept         CHAR(30)
                         REFERENCES departamentos
                         NOT NULL,
     titulación     CHAR(30),
     tipoContrato   CHAR(10),
     cargo          CHAR(30) NOT NULL,
     complemento    NUMERIC(5,2)
);
```

```
CREATE TABLE administrativos(
     DNI            NUMERIC(8) PRIMARY KEY,
     Nombre         CHAR(30) NOT NULL,
     Apellidos      CHAR(30) NOT NULL,
     FechNac        DATE NOT NULL,
     nivel          CHAR(30)
);

CREATE TABLE secretarios(
     DNI            NUMERIC(8) PRIMARY KEY,
     Nombre         CHAR(30) NOT NULL,
     Apellidos      CHAR(30) NOT NULL,
     FechNac        DATE NOT NULL,
     nivel          CHAR(30)
);

CREATE TABLE secre_depar(
     DNI       NUMERIC(8) REFERENCES secretarios
               ON DELETE CASCADE ON UPDATE CASCADE,
     NombreDept CHAR(30) REFERENCES departamentos
               ON DELETE CASCADE ON UPDATE CASCADE,
     PRIMARY KEY (DNI, NombreDept)
);
```

Hemos puesto en negrita los atributos heredados, que como vemos hay que replicar cada vez que especializamos.
Como los atributos *nombre* y *apellido* los asumimos como no nulos en las dos soluciones anteriores (secciones 9.4.2 y 9.4.3), en esta solución también lo hacemos así, y por tanto la restricción NOT NULL hay que llevarla a todas las tablas donde estos dos atributos vuelvan a declararse.
Como vemos si quisiéramos obtener un listado con todos los empleados, tendríamos que hacer una *unión* entre empleados, administrativos, secretarios, profesores y cargo.
Si hubiese discriminante, bajo esta solución podría omitirse su paso a relacional, pues el mero hecho de ser fila de una especialización ya supone un valor en el discriminante.
Si la ISA hubiese sido solapada, y hubiésemos permitido profesores que a la vez fuesen administrativos tendríamos algunos cambios:

1. Crear una tabla de *Profesores-Administrativos (no secretarios)* con todos los campos de ambos, más los de *empleados.*
2. Crear otra tabla de *Profesores-Secretarios* con todos los campos de ambos, más los de *empleados* y *administrativos.*

3. Crear otra tabla de *Cargos – Administrativos (no secretarios)* con todos los campos de ambos, más los de *empleados* y *profesores.*
4. Crear otra tabla de *Cargos – Secretarios* con todos los campos de ambos, más los de *empleados, administrativos* y *profesores.*

Aunque esto supone una solución, lógicamente no es la ideal cuando tenemos varias especializaciones y varios niveles en la jerarquía ISA, dada la cantidad de tablas que genera la explosión combinatoria de tantos casos.

Quizás la parte más problemática de esta solución son la implementación de las interrelaciones con la generalización. Hay situaciones que fuerzan una representación SQL compleja, la cual se hace inadmisible a medida que crecen el número de especializaciones.

Por ejemplo, supongamos que los empleados de la universidad se relacionasen M:N con una entidad *cuentas bancarias.* Sería difícil utilizar la típica tabla que generan las interrelaciones M:N para relacionar *empleados* con *cuentas* porque ahora la clave ajena con los empleados no es siempre el DNI de la tabla de *empleados*: unas veces sí que lo será, pero otras veces será el DNI de la tabla *profesores*, otra el de la tabla *administrativos* etc.. Quizás la solución más sencilla sea utilizar varias tablas N:M (e.g, una de *empleados* con *cuentas*, otra de *profesores* con *cuentas*, otra de *administrativos* con *cuentas*, otra de una de *cargos* con *cuentas*, y otra de *secretarios* con *cuentas*. Cuantas más especializaciones, más tablas.

Esto no sólo ocurre en las N:M sino en las 1:N en las que la clave ajena haga REFERENCES contra una generalización. Por ejemplo, una entidad *alumnos* relacionada con *profesores* con la interrelación *ser-tutor-personal-de*, que asigna a cada alumno un profesor tutor, generaría dos claves ajenas en la tabla de alumnos, ya que los *profesores* son generalización de los *cargos*, y necesitaríamos una clave ajena para tutores que no tuvieran cargo, y otra para tutores que sí lo tuvieran, y además comprobar que no pudiesen tener las 2 simultáneamente un valor no nulo. O bien, alternativamente tendríamos una tabla de alumnos tutorados por profesores con cargo y otra para los alumnos tutorados por profesores sin cargo, lo cual parece disparatado, y todavía lo es más a medida que tenemos más especializaciones y más interrelaciones que fuercen estas complicaciones.

9.4.4 ¿Cuál es la mejor solución?

Dependerá de cada caso. Habrá que valorar en cada caso concreto las ventajas y los inconvenientes. A veces, no llegaremos a una solución que salve todos los problemas, pero debemos de intentar pensar en cual es la *menos mala*.

9.4.4.1 Ventajas de tratar las ISAs como una interrelación 1:1

1. No fomenta la aparición de valores nulos, a diferencia de la solución de una sola tabla. Esto parece adecuado si el número de atributos de las especializaciones es grande frente al número de atributos de la generalización, pues son esos atributos los que se harían nulos con frecuencia en la solución de una sola tabla.
2. Es fácil de implementar una clave primaria común a todos los elementos de la jerarquía (e.g.,. el DNI). Es decir, está controlado que no haya dos empleados con el mismo DNI sea cual sea su especialización.
3. Controlamos que las interrelaciones con las especializaciones no afecten erróneamente a ningún elemento de la generalización. Por ejemplo, hemos evitado fácilmente que los administrativos que no fuesen secretarios participasen en la interrelación con *departamentos*.
4. Las interrelaciones con la generalización también llegan a una implementación sencilla que controla todos los casos permitiendo que se hereden las interrelaciones. Por ejemplo, *profesores* es generalización de *con-cargo*, al relacionarse *profesores* con *departamento* mediante una clave ajena, los profesores con cargo

indirectamente también se relacionan indirectamente con *departamentos* están al tener una clave ajena con *profesores*.

9.4.4.2 Inconvenientes de tratar las ISAs como una interrelación 1:1

1. La recuperación de toda la información de una especialización exige el *join* con todos sus ancestros (generalizaciones) en la jerarquía, lo que la hace más lenta. Por ejemplo, para obtener toda la información de un profesor hay que hacer

```
SELECT empleados.*, profesores.*
FROM empleados natural join profesores
WHERE nombre = 'Pepito' AND apellidos='Pérez';
```

2. Para insertar un elemento de la especialización tienes que insertar **antes** **todos** los ancestros de la jerarquía para sí no violar las claves ajenas entre especializaciones y generalizaciones. Por ejemplo, para insertar un profesor con cargo, primero hay que insertar una fila en *empleados*, con lo que ya podrías insertar la fila correspondiente en *profesores*, y finalmente, una vez has insertado el profesor ya puedes insertar la fila en *cargos*.
3. Como ya se ha indicado, no se controla el no-solapamiento, ni que sea total o parcial. Por ejemplo:
    1. Nada nos impide insertar una fila de administrativo con el DNI de un profesor, luego por lo tanto no podemos controlar que la ISA sea exclusiva.
    2. Si la ISA de empleados de universidad hubiese sido total, nada nos impide añadir un DNI en la tabla de empleados que no fuese ni de un administrativo, ni de un profesor, luego no podemos controlar que sea total.

9.4.4.3 Ventajas de la solución de una sola tabla

1. Es fácil de implementar una clave primaria común a todos los elementos de la jerarquía (p.e. el DNI). Es decir, está controlado que no haya dos empleados con el mismo DNI sea cual sea su especialización.
2. La recuperación de toda la información de una parte de la jerarquía es rápida y simple (no exige *joins* ni uniones). Basta una SELECT que filtre por los valores del discriminante que nos interesen.
3. Para insertar un elemento de la especialización tienes que insertar una única fila, no como ocurría en la solución 1.
4. Se puede controlar fácilmente que la ISA sea total. En nuestro ejemplo basta con quitar el caso 'E' en el discriminante. Y habría que simplificar el CHECK quitando el término correspondiente a Tipo CHECK(... OR (tipo='E' AND...).
5. Se puede controlar fácilmente que sea exclusiva. En el ejemplo, el discriminante no deja que un empleado pueda ser a la vez profesor y administrativo, porque el campo tipo sólo puede tomar un valor de los cuatro posibles, y ninguno de ellos está asociado a profesores que sean a la vez administrativos. Si la ISA hubiese sido solapada, y hubiésemos permitido profesores que a la vez fuesen administrativos tendríamos algunos cambios, como ya se indicó en la sección 9.4.2.
6. La propagación a las especializaciones de las interrelaciones con la generalización llegan a una implementación sencilla que se controla en todos los casos (e.g, la interrelación con una hipotética entidad de *provincias* de la que se habló en la sección 9.4.2. y cómo se propagaba el NOT NULL a las especializaciones si fuese necesario).

9.4.4.4 Inconvenientes de la solución de una sola tabla

1. Fomenta la aparición de valores nulos. Esto es porque cada especialización aporta sus atributos, los cuales estarán vacíos cuando la fila no se corresponda con dicha especialización. Este inconveniente hace menos adecuada esta solución si el número de atributos propios de las especializaciones es grande frente al número de atributos de la generalización.
2. No controlamos que las interrelaciones con las especializaciones no afecten a ningún elemento de la generalización. Por ejemplo, no hemos controlado que la pertenencia a departamentos no afectase a administrativos que no fuesen secretarios.

9.4.4.5 Ventajas de hacer una tabla independiente por especialización (+)

1. La recuperación de toda la información de una especialización no es tan rápida y simple como cuando trabajamos con una sola tabla, pero sigue siendo ágil y trivial (no exige *joins*). Por ejemplo, para obtener el nombre de todos los profesores tengan o no cargo, haríamos

```
SELECT DNI, nombre, apellidos FROM profesores
UNION
SELECT DNI, nombre, apellidos FROM cargos;
```

   Las uniones se pueden acelerar con UNION ALL para que la base de datos no pierda el tiempo en quitar los repetidos, pues suponemos que el mismo empleado no va a estar en repetido en varias tablas
   Nota: si bien, como veremos en la primera de las desventajas, esto no lo podemos asegurar desde las restricciones declaradas con CREATE TABLE.
2. Para insertar un elemento de cualquier especialización basta insertar una única fila.
3. Se puede controlar fácilmente que la ISA sea total, basta quitar la tabla de la generalización.
7. Se puede controlar que sea exclusiva, pues cada elemento sólo puede pertenecer a una sola de las tablas. En el ejemplo un empleado no puede ser profesor y administrativo simultáneamente, porque sólo puede aparecer en una de las tablas[16]. Si la ISA hubiese sido solapada, y hubiésemos permitido profesores que a la vez fuesen administrativos habría que haber creado 4 tablas nuevas para las 4 categorías híbridas que aparecen, como se discutió casi al final de la sección 9.4.3.
8. Como en la solución 1, evita también los valores nulos, pues cada tabla sólo tiene los campos que usan todas sus filas.
9. También controlamos que las interrelaciones con las especializaciones no afecten a ningún elemento de la generalización. Por ejemplo, hemos controlado que la pertenencia a departamentos no afectase a administrativos que no fuesen secretarios.

9.4.4.6 Inconvenientes de hacer una tabla independiente por cada especialización (+)

1. No es posible implementar una clave primaria común a todos los elementos de la jerarquía (e.g.,. el DNI). Es decir, no está controlado que no haya dos empleados con el mismo DNI sea cual sea su especialización. El mismo DNI *x* lo puede tener a la vez un profesor y un administrativo, sin embargo al ser filas de tablas diferentes, son personas diferentes (e.g., no tienen por qué llamarse igual, pues tampoco es controlable).

   Nota: Este tipo de controles podrían delegarse en ASSERTIONs[17], *triggers* y programas de aplicación

16 Se supone que hemos controlado (quizás vía el programa de aplicación que gestiona los datos) que no haya profesores con el mismo DNI que los administrativos (ver nota en primera de las desventajas).
17 En el Glosario parece una explicación detalla de lo que es uno objeto ASSERTION, con el correspondiente ejemplo.

que intervengan en las inserciones y modificaciones de DNIs, pues podrían lanzar un SELECT para comprobar si ese DNI ya existe en alguna de las tablas, y así prohibir por programa su utilización. Pero, la utilización de programa de aplicación, como ya hemos discutido alguna vez en otros escenarios (sección 6.2.2), no es tampoco una solución ideal, ya que cada vez que se haga un nuevo programa que intervenga en estas tareas tendré que volver recordar que he de comprobar primero si el DNI ya existe. Por el contrario, en las soluciones anteriores, la base de datos hace esa comprobación siempre, descargando a los programadores.

2. Las relaciones con la generalización hay situaciones que fuerzan una representación SQL compleja, la cual se hace inadmisible a medida que crecen el número de especializaciones (ver el final de la sección 9.4.3).

9.4.4.7 Seleccionar una solución híbrida (+)

A veces, en una misma jerarquía ISA que tenga más de un nivel de profndidad podemos utilizar varias de las 3 soluciones propuestas simultáneamente. Por ejemplo, y aunque en este caso no sea mucho mejor solución que quizás la "Solución 1: Tratar las ISAs como una interrelación 1:1", este es el resultado de hacer dicha solución 1 sólo a nivel de *empleados*, y a nivel de *secretarios*; mientras que aplicamos la solución *todo en una tabla* a nivel de *cargos*:

```
CREATE TABLE departamentos(
     Nombre    CHAR(30) PRIMARY KEY,
     Tfono     NUMERIC(9)
);

CREATE TABLE empleados(
     DNI       NUMERIC(8) PRIMARY KEY,
     Nombre    CHAR(30) NOT NULL,
     Apellidos CHAR(30) NOT NULL,
     FechNac   DATE NOT NULL
);

CREATE TABLE profesores(
     DNI       NUMERIC(8) PRIMARY KEY,
     Nombre         CHAR(30) NOT NULL,
     Apellidos CHAR(30) NOT NULL,
     FechNac        DATE NOT NULL,
     tieneCargo     BOOLEAN NOT NULL,
     NombreDept     CHAR(30) REFERENCES departamentos
                     NOT NULL,
     titulación     CHAR(30),
     tipoContrato   CHAR(10),
```

```
     cargo          CHAR(30),
     complemento    NUMERIC(5,2),
     CHECK (   tieneCargo AND cargo IS NOT NULL
           OR
          NOT tieneCargo
          AND (cargo IS NULL AND complemento IS NULL)
          )
);

CREATE TABLE administrativos(
     DNI       NUMERIC(8) PRIMARY KEY,
     Nombre         CHAR(30) NOT NULL,
     Apellidos CHAR(30) NOT NULL,
     FechNac        DATE NOT NULL,
     nivel          CHAR(30)
);

CREATE TABLE secretarios(
     DNI  NUMERIC(8) PRIMARY KEY
               REFERENCES administrativos
               ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE secre_depar(
     DNI       NUMERIC(8) REFERENCES secretarios
               ON DELETE CASCADE ON UPDATE CASCADE,
     NombreDept     CHAR(30) REFERENCES departamentos
               ON DELETE CASCADE ON UPDATE CASCADE,
     PRIMARY KEY (DNI, NombreDept)
);
```

IV. Resumen

En este tema hemos repasado el diagrama E/R visto en la asignatura de *Ingeniería del Software*, y hemos ampliado las capacidades del diagrama con nuevos elementos (i.e., interrelaciones de grado superior, agregaciones y interrelaciones ISA).
Las interrelaciones reflexivas, por dar tradicionalmente muchos quebraderos de cabeza a los alumnos, se han reforzado con secciones que las han ligado a la representación de algunas estructuras de datos convencionales.
Se ha justificado la necesidad de las interrelaciones *n-arias* para recoger restricciones que luego se transforma en claves relacionales, y que no pueden ser representadas de otra forma en el modelo. Se ha incidido en la importancia de las interrelaciones de grado superior para la representación de datos con dimensión temporal. Hemos introducido las agregaciones, y las hemos diferenciado claramente de las interrelaciones de grado superior.
A mayores, el tema se ha enfocado en todo momento a entresacar toda la semántica de los diagramas y plasmarla en los CREATE TABLE de SQL. Las restricciones de los CREATE TABLE permiten capturar la mayor parte de la semántica del diagrama, pero hemos visto que en ocasiones no son suficiente, casi todas esas ocasiones están ligadas al mantenimiento de cardinalidades mínimas "1" (e.g., cardinalidades mínimas 1 en los dos extremos, cardinalidad mínima 1 en la parte "a varios"). Estos casos hemos visto que también pueden controlarse, pero con otro tipo de técnicas que se verán en asignaturas posteriores.
Para pasar los diagramas E/R a tablas SQL hemos visto una serie de reglas que hemos razonado con gran cantidad de ejemplos, para que el alumno no sólo se las aprenda, sino qué sepa el porqué de cada una y eso le facilite utilizar el sentido común a la hora de saber qué regla y hasta dónde quiere llegar en su implementación en cada caso.
Se ha dedicado la parte final del tema a la revisión de técnicas para pasar a SQL las interrelaciones ISA. La semántica de estas interrelaciones no está presente de una forma natural en el modelo relacional, por eso hemos discutido varias técnicas, sus pros y sus contras.

V. Glosario

AIA. Atributo Identificador Alternativo. En la terminología de [de Miguel y Piattini 1993] representa un atributo o conjuntos de atributos que podrían, alternativamente, haber sido elegidos como AIP o clave de una entidad. Es un concepto paralelo al de clave candidata no primaria en el modelo relacional.
AIP. Atributo Identificador Principal. En la terminología de [de Miguel y Piattini 1993] representa la clave de una entidad por la que se identifica cada elemento de la misma. Es un concepto paralelo al de clave primaria en el modelo relacional.
ASSERTION o aserto. En SQL un aserto es un objeto de la base de datos, por lo que tiene su propia sentencia CREATE (se crea con CREATE ASSERTION), y hace el mismo papel que una restricción CHECK, sólo que en un CHECK sólo se pueden comparar los valores que en una misma fila toman campos de la tabla en la que están definidos, pero en un aserto podemos hacer comparaciones en la que intervengan campos de distintas tablas y/o de distintas filas. Para ello, en el aserto se pueden definir un predicado con comparaciones sobre una o varias subconsultas, y se verifica si el resultado de ese predicado con subconsultas es el esperado. Debido a que las subconsultas asociadas al aserto se ejecutarían cada vez que hiciésemos un INSERT/DELETE/UPDATE sobre alguna

de las tablas que intervienen en esas subconsultas, los asertos tienden a ralentizar el sistema, y por eso, a pesar se ser una característica estándar de SQL desde 1992, los fabricantes de SGBDs no los incluyen en sus implementaciones. Veamos un ejemplo inspirado en el que con una aserción controlamos que todo autor ha de escribir al menos un libro (i.e., controlamos el "1" en extremo (1,n) de la interrelación N:M *autores (0,n) ← escriben → (1,n) libros*:

```
CREATE ASSERTION autores_escribe_1_libro
CHECK ( NOT EXISTS (
            SELECT cod_autor FROM autores
            EXCEPT
            SELECT cod_autor FROM escribe));
```

como vemos, es muy ineficiente tener que ejecutar esa consulta compleja cada vez que hagamos operaciones de INSERT/DELETE/UPDATE sobre las tablas *escribe* y *autores.*

Cardinalidad. La cardinalidad de una interrelación es una restricción por la que definimos el número de elementos de una entidad que se relacionan con un elemento de otra. La cardinalidad puede ser mínima o máxima, la primera puede valer "0" ó "1", y define si cada elemento de la entidad participa de forma obligatoria o no en la interrelación, por lo que se conoce también como *restricción de participación* [Connolly y Begg 2005]; y además cuando vale "1" es sinónimo de la *dependencia por existencia* (i.e., una entidad no puede existir sin participar en esa interrelación). Algunos textos, por tanto, no consideran cardinalidad a la cardinalidad mínima, sino que la consideran restricción de participación, o restricción de dependencia por existencia. Sin embargo, todos los textos consideran cardinalidad a la cardinalidad máxima, que puede valer "1" o "n" y que representa el número de veces que como máximo puede participar un elemento de una entidad en una interrelación.

Commit. Sentencia SQL que indica el final exitoso de la transacción en curso y, a la vez, el comienzo de la siguiente transacción. Mediante una transacción, en una sesión SQL podemos agrupar varias operaciones SQL, de manera que el resto de usuarios o sesiones no verían el resultado de las mismas hasta que terminásemos dicha transacción ejecutando *commit;* pero si por el contrario, nos arrepentimos de esos cambios y queremos deshacerlos antes de que los vean otros usuarios terminaríamos la transacción ejecutando la sentencia antagónica a *commit*, que es la sentencia *rollback*.

Herramienta CASE. Programa informático que sirve para dar soporte informático a las tareas (normalmente de diseño pero puede abarcar más aspectos), asociadas al ciclo de vida de un proyecto software. CASE son las siglas de *Computer Aided Software Engeneering* (i.e., Ingeniería del Software Asistida por Computador). En proyectos software que integran una base de datos, las herramientas CASE típicamente permiten diseñar el esquema de base de datos, normalmente, valiéndose de algún tipo de representación gráfica del mismo y formularios que permiten completar esa información gráfica, generando así internamente una metabase o diccionario de datos. Con todos esos datos del diccionario, la mayoría de las herramientas CASE son capaces de generar un esquema lógico, por ejemplo en SQL.

Nullable. Dícese de un campo que puede tomar el valor nulo.

Trigger o disparador. Es la implementación SQL de una regla ECA (acrónimo de *Evento Condición Acción*). En una regla ECA definimos que cuando ocurra un determinado *Evento* (e.g., INSERT/DELETE/UPDATE de unos determinados campos en una determinada tabla), y además se cumpla una determinada *Condición* (e.g., el valor nuevo de un campo sea mayor que tal o cual, etc …), se ejecute una determinada *Accion* (e.g., abortar la transacción con un mensaje de error, calcular el valor de otro campo etc …).

VI. Bibliografía

Referencias:

[Celko 2005] Celko, J. ***SQL for Smarties 3rd Ed.***. Morgan Kaufmann, 2005.

(Disponible en acceso abierto en https://datubaze.files.wordpress.com/2015/03/celkos-sql-for-smarties-2005.pdf).

[Chen 1976] Chen, Peter P. ***The Entity/Relationship Model: Toward a Unified View of Data***. ACM Transactions on Database Systems (TODS). Vol. 1(1), 1976. pp 9-36.

(Disponible en acceso abierto en http://www.inf.unibz.it/~nutt/IDBs1011/IDBPapers/chen-ER-TODS-76.pdf).

[Connolly y Begg 2005]. Connolly, Thomas M. y Begg, Carolyn E. ***Sistemas de Bases de Datos 4ª ed***. Pearson, 2005.

(Disponible en UBUvirtual: http://0-www.ingebook.com.ubucat.ubu.es/ib/NPcd/IB_BooksVis?cod_primaria=1000187&codigo_libro=2155)

[Costal-Costa 2005] Costal Costa, Dolors ***Módulo didáctico 4: Introducción al diseño de bases de datos***. En Software Libre. Bases de Datos. *Rafael Camps Paré, Luis Alberto Casillas Santillán, Dolors Costal Costa, Marc Gilbert Ginestà, Came Martín Escofet, Oscar Pérez Mora*. UOC Formación de Postgrado. 2005

[Date 2000] Date, Chris J. ***Introduction to Database Systems 7th ed.*** Addison Wesley 2000.

[de Miguel y Piattini 1993] de Miguel, Adoración y Piattini, Mario. ***Concepción y Diseño de Bases de Datos. Del Modelo E/R al Modelo Relacional***. RA-MA, 1993.

[Eisenberg et *al.* 2004] Eisenberg, Andrew, Melton, Jim, Kulkarni, Krishna, Michels, Jan-Eike y Zemke, Fred. ***SQL:2003 Has Been Published.*** ACM SIGMOD Vol 33(1), 2004 pp 119-126.

(Disponible en acceso abierto en http://sigmod.org/publications/sigmod-record/0403/E.JimAndrew-standard.pdf).

[Elmasri y Navathe 2007] Elmasri, Ramez y Navathe, Shamkant B. ***Fundamentos de Sistemas de Bases de Datos 5ª ed***. Pearson, 2007

(Disponible en UBUvirtual: http://0-www.ingebook.com.ubucat.ubu.es/ib/NPcd/IB_BooksVis?cod_primaria=1000187&codigo_libro=2886)

[Fowler 2003] Fowler, Martin. ***UML distilled. A brief guide to the standard object modelling 3rd Ed.*** Addison Wesley 2003.

(Disponible en acceso abierto en http://www.saeedsh.com/resources/UML%20Distilled%203rd%20Ed.pdf).

[Gardarin 1993] Gardarin, G., ***Dominar las Bases de Datos***, Ediciones Gestión 2000, 1993.

[METRICA V3 2001] Ministerio de Administraciones Públicas, ***Metodología MÉTRICA Versión 3: Técnicas y Prácticas***. 2001. Documento electrónico en http://administracionelectronica.gob.es/pae_Home/dms/pae_Home/documentos/Documentacion/Metodologias-y-guias/Metricav3/METRICA_V3_Tecnicas.pdf , última visita 22-08-2015.

[McAllister 1995] McAllister A ***Modeling N-ary Data Relationships in CASE Environments***, En Procs. of the 7th International Workshop on Computer Aided Software Engeneering, Toronto, Canda, 1995, pp 132 – 140.

[Silberschatz et. al 2006] Silberschatz, Abraham, Korth, Henry F. y Sudarshan, S. ***Fundamentos de Bases de Datos 5ªEd.*** McGraw-Hill, 2006.

(Disponible en UBUvirtual: http://0-www.ingebook.com.ubucat.ubu.es/ib/NPcd/IB_BooksVis?cod_primaria=1000187&codigo_libro=4356).

[Tsichritzis y Klug 1978] Tsichritzis, Dennis y Klug, Anthony ***The ANSI/X3/SPARC DBMS framework report of the study group on database management systems. Information Systems***, Vol. 3(3), 1978, pp 173-191.

[Ullman y Widom 1997] Ullman, Jeffrey D. y Widom, Jennifer. ***A First Course in Database Systems***. Prentice Hall, 1997.

Bibliografía comentada:

El modelo E/R es un tema de prácticamente cualquier libro de introducción a las Bases de Datos (e.g., capítulo 12 de [Date 2000], capítulos 8 y 9 de [de Miguel y Piattini 1993], capítulos 3 y 4 de [Elmasri y Navathe 2007], capítulo 6 de [Silberschatz et. al 2006], secciones 2.2, 2.4.3, 2.4.4 y 2.6 en [Ullman y Widom 1997]). Los capítulos 11 y 12 de [Connolly y Begg 2005] son también muy completos, pero lamentablemente el ejemplo que utiliza no está traducido al español y puede resultar costoso de seguir si no se está al corriente de la terminología inglesa de ese ejemplo, y el uso de UML en lugar de diagramas E/R terminan de complicarlo.

Sin embargo, el enfoque que se ha impartido en estos apuntes es muy distinto al que hay en los libros, en tanto que por dar un enfoque práctico, nos hemos centrado en la representación SQL de las restricciones del modelo E/R, y en ese enfoque la literatura no acostumbra a extenderse tanto como nos hemos extendido en los apuntes. Quizás el capítulo 23 de [de Miguel y Piattini 1993] sea la referencia con un enfoque más similar al pretendido; incluso el capítulo 13 en [Date 2000] apuntaría en la misma dirección que nosotros pero de una forma mucho más escueta.

Cada autor de bases de datos tiende a tomarse ciertas licencias en cuanto a la simbología usada. Las disparidades se concentran en cómo representar las dependencias, las cardinalidades y determinadas características de la ISA. Nosotros para tomar una simbología normalizada en España hemos decidido adoptar la que viene en el capítulo del *Modelo Entidad/Relación Extendido* de [METRICA V3 2001]. La sección 6.7.6 de [Silberschatz et. al 2006] hace una interesante reseña sobre el tema de las distintas simbologías.

Las reglas de paso de un modelo E/R a tablas, sin entrar en su especificación SQL, puede encontrarse prácticamente en todos los textos, incluso desde la aparición del modelo en [Chen 1976]. (e.g., el capítulo 16 de [Connolly y Begg 2005], capítulo 13 de [Date 2000], [de Miguel y Piattini 1993] capítulo 23, el capítulo 7 de [Elmasri y Navathe 2007], la sección 6.9 de [Silberschatz et. al 2006], o la sección 3.3 de [Ullman y Widom 1997]), pero casi ninguno entra en ternarias, agregaciones, y la ISA no la dan la amplitud de tratamiento que aquí se ha dado. Así:

- La sección 28.1 de [Celko 2005] discute de forma más avanzada aún que en los apuntes las estructuras de árbol vistas dentro del apartado de interrelaciones reflexivas, y la sección 30.1 apenas añade más a lo dicho en los apuntes en cuanto a creación de tablas para grafos, si bien esta referencia es excelente para quien quiera adentrarse en cómo hacer consultas sobre este tipo de tablas.
- Las cardinalidades en las interrelaciones *n*-arias vienen explicadas brevemente en la sección 3.9.2 de [Elmasri y Navathe 2007]. La explicación de la sección 11.6.4 de [Connolly y Begg 2005], hace un análisis

de las cardinalidades basado en UML que no coincide con el nuestro [McAllister 1995]. El paso a tablas en [Elmasri y Navathe 2007] elude la discusión sobre las claves candidatas resultantes de una interrelación *n*-aria, como hacen casi todos los autores de textos básicos. [McAllister 1995] es la referencia que se recomienda para profundizar sobre el significado de las cardinalidades, y [Costal-Costa 2005] sección 3.4 por su claridad en lo referente a su paso a tablas relacionales.

- La agregación no es soportada por demasiados textos. Sí que lo hace de manera breve [Silberschatz et. al 2006] sección 6.7.5 más el paso a tablas en la 6.9.6. La agregación que aparece en la sección 12.2 de [Connolly y Begg 2005] y en la sección 4.7.4 de [Elmasri y Navathe 2007]es la de UML, por lo que es un concepto totalmente distinto del que aquí se ha tratado.
- La dimensión temporal del modelo E/R se discute en la sección 9.3 de [de Miguel y Piattini 1993], si bien los planteamientos de fondo son correctos, su interpretación confusa o excesivamente relajada de las interrelaciones ternarias y sus cardinalidades puede inducir a error al alumno.
- El la sección 9.2 de [de Miguel y Piattini 1993] hace un tratamiento muy similar al nuestro de las interrelaciones ISA en cuanto a su división en totales/parciales y exclusivas/solapadas. También se usan discriminantes. [Connolly y Begg 2005] en la sección 12.1.6 llama *restricciones de participación* a la dicotómia *total/parcial* y *restricciones de disyunción* a la dicotomía *exclusiva/solapada*. En la sección 4.3 de [Elmasri y Navathe 2007], aunque no coinciden los símbolos para la ISA, utiliza igualmente esa división entre totales/parciales y exclusivas/solapadas.
- Las 3 opciones de transformación de la ISA pueden encontrarse en el capítulo de *Reglas de Transformación* de [METRICA V3 2001], aunque de manera muy escueta. En la sección 23.4.9 y 23.4.9 de [de Miguel y Piattini 1993] pueden encontrarse más desarrolladas, incluyendo el SQL necesario para trabajar con el CHECK en la solución de una única tabla. También se presentan :
  - las tres soluciones en la sección 16.1(6) de [Connolly y Begg 2005], pero su opción 1 es nuestra solución 2, sus opciones 2 y 3 coincide con nuestra solución 3 sólo que implementadas en dos escenarios de totalidad-exclusividad diferentes. Finalmente, la opción 4 es nuestra solución 1.
  - las tres soluciones en [Elmasri y Navathe 2007], la opción que llama *8A* es nuestra solución 1, la *8B* es nuestra solución 3 y las *8C* y *8D* son variantes de nuestra solución 2, sólo que en *8C* utiliza un único atributo como discriminante, justo como nosotros, pero en la *8D* para dar un soporte elegante a las ISAs solapadas, utiliza *n* discriminantes booleanos para significar *n* especializaciones.
  - las soluciones que hemos numerado como 1 y 3 en la sección 6.9.5 de [Silberschatz et. al 2006].
  - y finalmente algunos autores sólo contemplan una de las 3 opciones vistas (e.g., [Date 2000] y [Ullman y Widom 1997] sólo contemplan nuestra solución 1).

VII. Material complementario

El modelado conceptual con otros modelos distintos al modelo E/R puede encontrarse en:

- Con UML:
  - [Connolly y Begg 2005] en el capítulo 11 usa íntegramente UML en lugar del modelo E/R, a pesar de que el título del capítulo no lo indica.

  - Una sección más abreviada al respecto son la 3.8 y 4.6 de [Elmasri y Navathe 2007], y finalmente,
  - ***DB9 Unified Modeling Language*** . Vídeos 1 y 2, *.Prof. Jennifer Widom, Standford University.* https://lagunita.stanford.edu/courses/DB/UML/SelfPaced/courseware/bb14e823816749bba8a30e1ebee5bb53/360b0e504a814a09bf060db8056819f0/

- Con ODL (*Object Definition Language*). que es el lenguaje DDL estándar[18] para la definición de bases de datos orientadas a objeto, el cual es estándar ODMG. [Ullman y Widom 1997], en su capítulo 2 (secciones 2.1, 2.4, 2.5) utiliza ODL, y en la sección 3.2 da las reglas de paso de ODL a relacional.
- Las cardinalidades de Merise en las interrelaciones de grado superior también vienen explicadas y discutidas en [McAllister 1995].

18 Es estándar ODMG (*Object Data Management Group*).

Licencia

Autor: Jesús Maudes Raedo
Área de Lenguajes y Sistemas Informáticos
Departamento de Ingeniería Civil
Escuela Politécnica Superior
UNIVERSIDAD DE BURGOS
2017

Este obra está bajo una licencia Creative Commons Reconocimiento-NoComercial-CompartirIgual 3.0 Unported. No se permite un uso comercial de esta obra ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula esta obra original
Licencia disponible en http://creativecommons.org/licenses/by-nc-sa/3.0/