Diseño lógico de bases de datos

PID_00270596

Xavier Burgués Illa
Josep Cuartero Olivera

Tiempo mínimo de dedicación recomendado: 6 horas

Universitat Oberta de Catalunya

Xavier Burgués Illa

Ingeniero informático, profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad Politécnica de Cataluña. Imparte las asignaturas de Bases de datos y de Programación en la Facultad de Informática de Barcelona y es consultor de la Universitat Oberta de Catalunya.

Josep Cuartero Olivera

Ingeniero informático y máster en Sociedad de la Información y del Conocimiento en la Universitat Oberta de Cataluña (UOC). Profesional informático en el sector privado ejerciendo de experto en sistemas de información. Es profesor colaborador de la UOC.

La revisión de este recurso de aprendizaje UOC ha sido coordinada por la profesora: Àngels Rius Gavidia

Quinta edición: febrero 2020

Av. Tibidabo, 39-43, 08035 Barcelona
Autoría: Xavier Burgués Illa, Josep Cuartero Olivera
Producción: FUOC

Índice

Introducción

El proceso de desarrollo de una base de datos para un sistema de información es un proceso secuencial formado por un conjunto de diferentes fases que nos acercan al resultado final de manera progresiva. Después de la etapa de diseño conceptual, que permite obtener una especificación basada en el esquema conceptual generado a partir de los requisitos y el análisis del dominio de la aplicación, tenemos que llevar a cabo el diseño lógico.

Empezaremos el estudio de este módulo revisando el resultado obtenido en la etapa previa, la de diseño conceptual. Este repaso hay que realizarlo antes de iniciar la siguiente etapa, la de diseño lógico, y va a servirnos como punto de partida para seguir con el proceso de diseño de bases de datos.

A continuación, nos centraremos en la etapa de diseño lógico y estudiaremos las pautas para llevarla a cabo. El diseño lógico tiene por objetivo conseguir un esquema lógico de la base de datos a partir del esquema conceptual que hemos obtenido en la fase anterior. Se puede considerar, pues, como la etapa que permite transformar un modelo conceptual en un modelo lógico de la futura base de datos.

En este material utilizaremos modelos conceptuales basados en el lenguaje UML y los transformaremos en modelos lógicos relacionales, denominados simplemente *modelos relacionales*.

En este módulo también veremos la teoría de la normalización, que permite asegurar que el esquema relacional satisface una serie de condiciones que garantizan una mejor calidad de la base de datos.

Objetivos

El contenido de estos materiales didácticos os permitirá alcanzar los objetivos siguientes:

1. Entender el diseño lógico como actividad de transformación.
2. Conocer las trampas de diseño e identificar las situaciones en las que se pueden producir.
3. Conocer los efectos negativos que puede suponer la existencia de valores nulos.
4. Conocer las alternativas de transformación de los elementos del modelo conceptual en los diferentes elementos del modelo relacional.
5. Conocer y aplicar los diferentes mecanismos de definición de restricciones en el modelo relacional.
6. Conocer las anomalías que se pueden producir en un esquema no normalizado.
7. Conocer las formas normales hasta la quinta, y ser capaces de aplicarlas a un esquema dado.

1. Introducción al diseño lógico

El diseño lógico es una etapa intermedia de las que componen el proceso de desarrollo de una base de datos. Por lo tanto, se parte de los resultados de una etapa previa (la etapa del diseño conceptual) y se producen otros nuevos, que a su vez servirán como punto de entrada de una etapa posterior (el diseño físico).

Si hablamos de desarrollo de software en general, la etapa de diseño lógico parte de las especificaciones del sistema para diseñar una solución independiente de la tecnología, que después se refinará y se implementará en etapas posteriores del desarrollo. Si nos centramos en la parte de datos del diseño lógico, partiremos de la parte de la especificación que corresponde a la modelización conceptual del dominio de la aplicación, para obtener un esquema de la base de datos expresado en un lenguaje correspondiente a algún modelo lógico de base de datos, pero sin adoptar una versión concreta de ningún sistema de gestión de base de datos (SGBD) ni entrar en detalles de optimización o refinamiento de la base de datos, que se dejarán para etapas posteriores de desarrollo.

Existen varias opciones para el lenguaje de modelización conceptual, en el que estará expresada la información de entrada al diseño lógico, y para el modelo lógico que utilizaremos para expresar la solución resultante de este diseño. En este material, utilizaremos modelos conceptuales basados en el lenguaje UML y los transformaremos en modelos lógicos relacionales.

Empezaremos el estudio de este módulo con el tratamiento de las denominadas *trampas de diseño*, que son errores que se pueden haber cometido al hacer el diseño conceptual. Es necesario asegurarse de que el modelo conceptual está libre de estos antes de tomarlo como punto de partida para iniciar el diseño lógico.

A continuación, nos centraremos en la etapa de diseño lógico propiamente dicha. Analizaremos un diagrama de clases de UML y veremos cuáles son las diferentes alternativas de transformación de cada uno de los elementos de dichos diagramas en elementos del modelo relacional. También estudiaremos las ventajas y los inconvenientes de cada alternativa para poder elegir de manera correcta, en caso de disponer de más de una alternativa.

En este módulo también veremos la teoría de la normalización. Entre otras aplicaciones, esta teoría permite poner un sello de calidad al diseño obtenido desde el punto de vista de la ausencia de redundancia, facilidad de mantenimiento de la consistencia y eficiencia en las operaciones de actualización de

la base de datos derivada del diseño lógico. Además, en el supuesto de que el diseño no satisfaga las propiedades requeridas, la teoría de la normalización incluye procedimientos de corrección del diseño.

2. Reconsideración del modelo conceptual: trampas de diseño

Reflexión

Hay diferentes tipos de diagramas para representar los modelos conceptuales, pero en este texto emplearemos los diagramas de clases del lenguaje UML.

En este apartado veremos algunos errores que se pueden cometer durante la elaboración del diseño conceptual de la base de datos. Estos errores están relacionados con conceptualizaciones erróneas de la realidad que se han incorporado en el modelo conceptual.

Precisamente por la naturaleza del diseño conceptual, que implica interpretar y modelizar conceptos del mundo real, surge la posibilidad de cometer errores dependiendo de si la interpretación que hacemos es del todo correcta o no. Es importante evitar estos errores, puesto que pueden provocar graves problemas en fases posteriores del desarrollo de la base de datos.

Algunos de estos errores son recurrentes y siguen unos patrones que nos permiten identificarlos; son lo que denominamos **trampas de diseño**. A continuación presentaremos cinco de estas trampas, puesto que si las conocemos nos será más sencillo detectarlas y evitar caer en ellas.

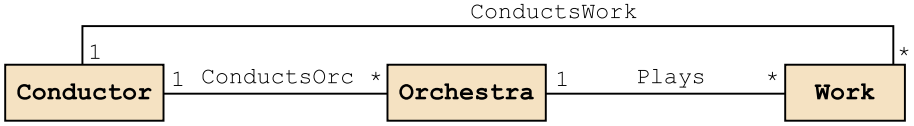
2.1. Abanico

El abanico es la primera trampa que se nos puede presentar cuando tenemos tres tipos de entidades relacionadas mediante tres tipos de relaciones binarias (1..*).

Decimos que hemos caído en la **trampa del abanico** cuando uno de los tipos de relación es derivado de los otros dos y, al no querer incorporar el derivado en el esquema conceptual, nos equivocamos y eliminamos uno de los que no lo es.

Ilustrémoslo con un ejemplo. Imaginemos que intentamos modelizar las grabaciones que hace una discográfica de música clásica. En la producción de esta compañía, cada obra (*Work*) es interpretada (*Play*) por una orquesta (*Orchestra*) una sola vez, y cada orquesta está dirigida siempre por un director (*Conductor*). La situación se puede modelizar como se indica en la figura 1.

Figura 1. Tres tipos de relaciones binarias, uno de los cuales es derivado



Este modelo, a pesar de que contiene los elementos (tipos de entidad y tipos de relaciones) necesarios, no es del todo correcto, porque falta una restricción: si una orquesta interpreta alguna obra y un director dirige dicha orquesta, el director dirige la obra. Es decir, el tipo de relación *ConductsWork* es un tipo derivado a partir de los otros dos.

Reflexión

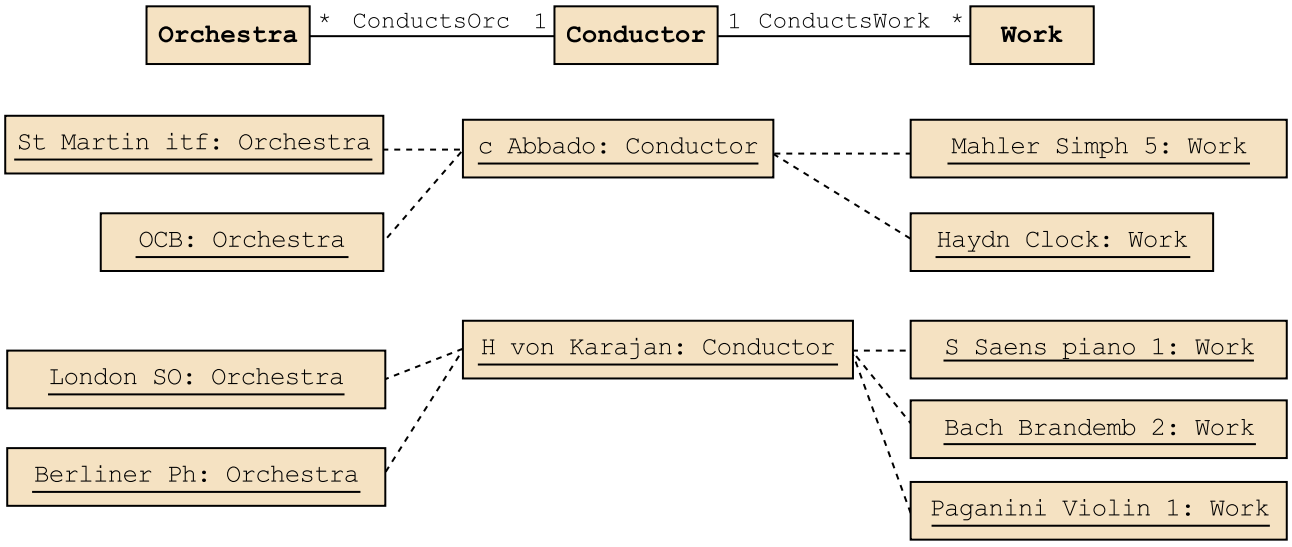
Cuando hay elementos derivados, podemos decidir eliminarlos del diagrama de clases si pensamos que de este modo queda más claro y el elemento derivado no es de interés para la base de datos que diseñamos.

Si decidimos representar el dominio con el esquema de la figura 2, habremos caído en la trampa, puesto que habremos sacado el tipo de relación *Plays* en lugar de eliminar *ConductsWork*.

También puede suceder que caigamos en la trampa directamente, sin ser conscientes de que incluimos un tipo de relación derivado y, en cambio, no incluyamos en el modelo el tipo de relación a partir del cual se puede derivar la nueva información.

Al caer en la trampa, dejamos de representar información relevante. En el ejemplo, no sabemos qué orquesta ha interpretado cada obra. El nombre *abanico* tiene su origen en la forma que resulta de representar las instancias y los vínculos entre ellas, como se muestra en la figura 2.

Figura 2. La trampa del abanico



El modelo conceptual correcto, sin representar el tipo de relación derivado, se puede ver en la figura 3.

Figura 3. Esquema correcto de la figura 1

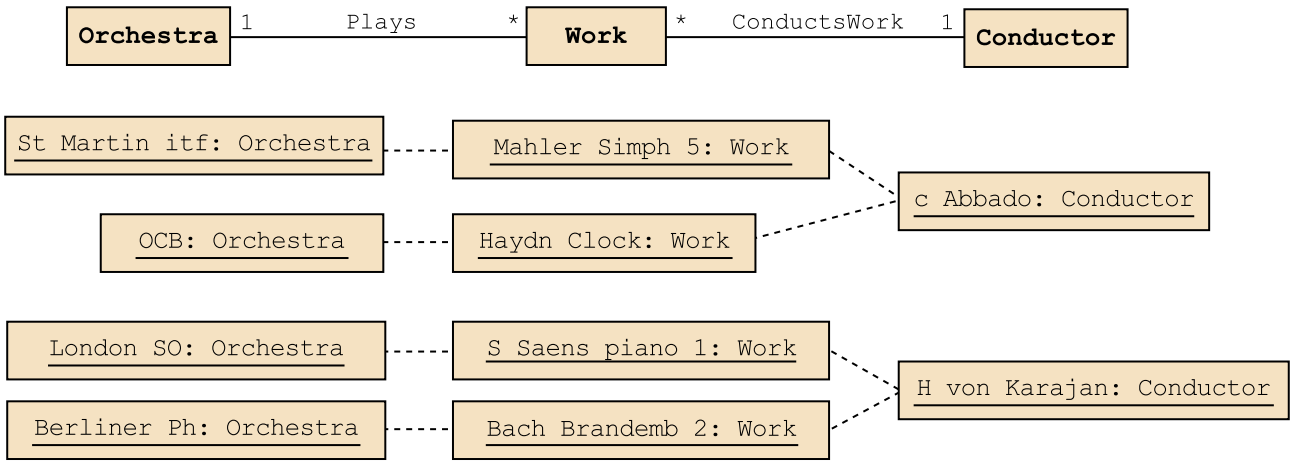
2.2. Corte

Esta otra trampa se presenta en la misma situación que la del abanico: cuando tres tipos de entidad están relacionados por tres tipos de relaciones binarias (1..*), uno de los cuales es un tipo derivado de los otros dos.

Decimos que hemos caído en la **trampa del corte** cuando, al no querer incorporar el tipo de relación derivado en el esquema conceptual, nos equivocamos y eliminamos uno de los que no lo es. En este caso, la consecuencia es que la información sobre la relación entre objetos de las entidades que no han quedado directamente relacionadas queda supeditada a la existencia de algún objeto de la tercera entidad para hacer de puente.

Continuando con el ejemplo anterior, podemos entender mejor la problemática de la trampa de corte y por qué esta trampa nos habría llevado al modelo conceptual de la figura 4.

Figura 4. La trampa del corte



El modelo de la figura 4 vuelve a ser incorrecto porque incluye un tipo de relación derivado (*ConductsWork*) y se olvida de uno que no lo es (*ConductsOrc*).

Si observamos las instancias que acompañan al modelo conceptual de la figura 4, no se ven las consecuencias que tiene caer en esta trampa. A pesar de esto, resulta muy fácil ver que este modelo no es correcto, si pensamos:

1) cómo podemos indicar quién es el director de una orquesta que todavía no ha interpretado ninguna obra, o

2) que si eliminamos una obra, perdemos datos del director de la orquesta que la interpretó.

De aquí proviene el nombre de esta trampa: eliminar una obra puede "cortar" la conexión entre una orquesta y su director.

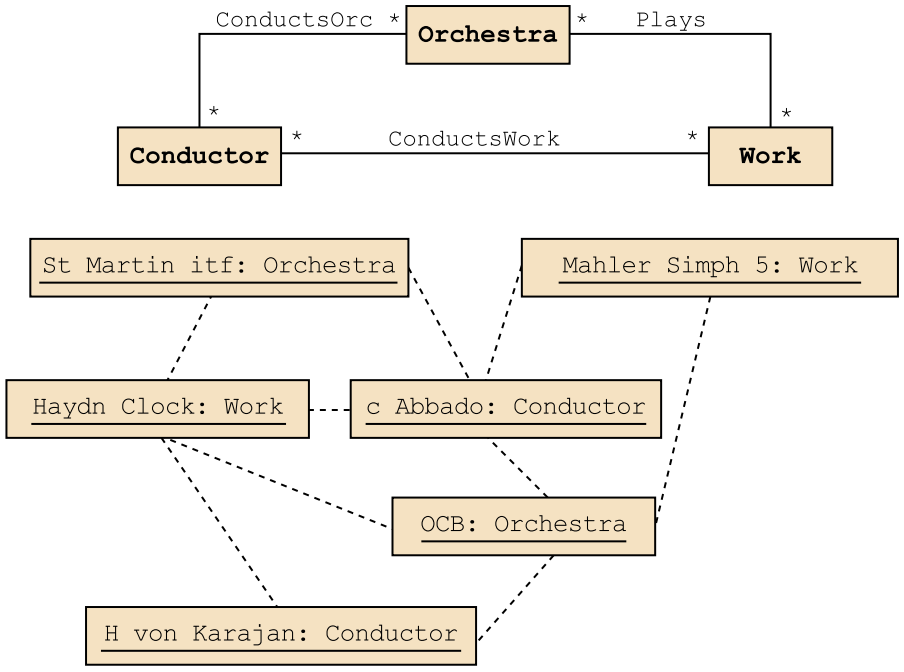
2.3. Pérdida de afiliación

La tercera trampa que podemos detectar es lo que se conoce como *pérdida de afiliación*. Se puede presentar cuando hay un tipo de relación ternaria y tres tipos de relaciones binarias entre los tipos de entidad que lo constituyen.

Decimos que hemos caído en la **trampa de pérdida de afiliación** cuando en vez de representar un tipo de relación ternaria, representamos los tipos de relaciones binarias que se derivan de ella.

Retomemos el ejemplo de los casos anteriores, pero ahora supongamos que una misma orquesta puede ser dirigida por varios directores y una misma obra puede ser interpretada por varias orquestas. Esta situación nos puede llevar a un modelo como el de la figura 5.

Figura 5. La trampa de la pérdida de afiliación



Si suponemos que en el mundo real modelizamos el hecho de que C. Abbado ha dirigido la orquesta de St. Martin in the Fields interpretando la Sinfonía del Reloj de Haydn, que H. von Karajan ha dirigido la OCB interpretando la Sinfonía del Reloj> y que C. Abbado ha dirigido la OCB interpretando la 5.ª sinfonía de Mahler, las instancias serán las que se ven en la figura 5. Sin embargo, a la vista de las instancias, podríamos pensar que Claudio Abbado ha dirigido la OCB interpretando la Sinfonía del Reloj, lo cual no es cierto. Es decir, el uso de múltiples tipos de relaciones binarias en lugar de una ternaria puede

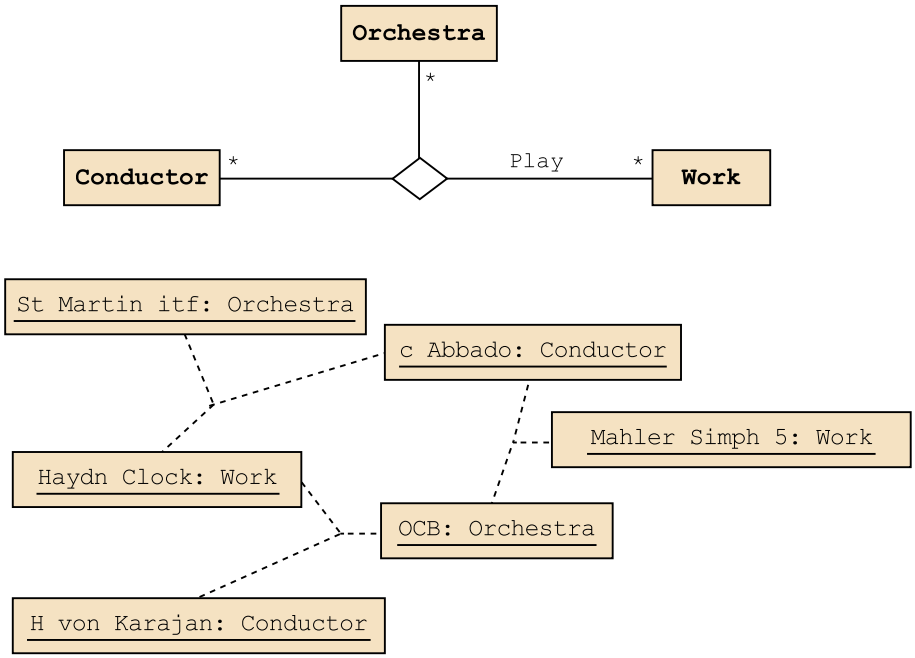
Reflexión

Aplicamos la misma denominación al caso en el que pretendemos representar esta situación con dos de los tipos de relaciones binarias. Por ejemplo, si solo modelizamos los tipos de relación *ConductsWork* y *Plays*.

provocar confusiones. Hemos perdido parte de la información que teníamos, puesto que sabemos qué directores han dirigido una orquesta, pero no cuáles de las obras interpretadas por la orquesta han sido dirigidas por cada director.

El modelo conceptual correcto que se ajusta a la situación planteada corresponde a un modelo con un tipo de relación ternaria, tal y como se describe en la figura 6.

Figura 6. Esquema conceptual correcto



2.4. Aridad de los tipos de relación

En el caso anterior hemos visto que una equivocación en el número de tipos de entidad que participan en un tipo de relación puede ser una fuente de errores en los esquemas conceptuales.

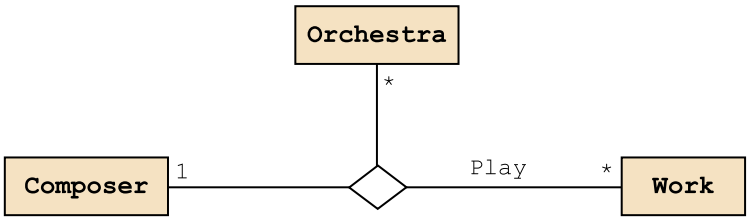
En este punto, veremos que incorporar un tipo de entidad a un tipo de relación que no le corresponde también puede ser fuente de errores en el diseño conceptual.

La **trampa de aridad de los tipos de relación** se puede presentar cuando hay un tipo de relación de grado mayor que 2 que tiene alguna multiplicidad de valor 1. En este caso, es posible que la entidad que se encuentra en el lado con multiplicidad igual a 1 en realidad no deba formar parte del tipo de relación. Decimos que hemos caído en esta trampa cuando hemos incluido la entidad de manera errónea.

Así pues, cuando nos aparece una multiplicidad igual a 1 en un tipo de relación ternaria (o de grado más alto), debemos tener mucho cuidado y verificar si el grado del tipo de relación es tal como lo planteamos o si en realidad se trata de dos o más tipos de relación de grado más bajo.

Pensemos ahora en un contexto en el que se quiere modelizar el hecho de que las obras son interpretadas por orquestas y que cada obra tiene un compositor, con la restricción de que cada obra tiene un único compositor. Un posible modelo conceptual para esta descripción se muestra en la figura 7.

Figura 7. La trampa de la aridad de los tipos de relación

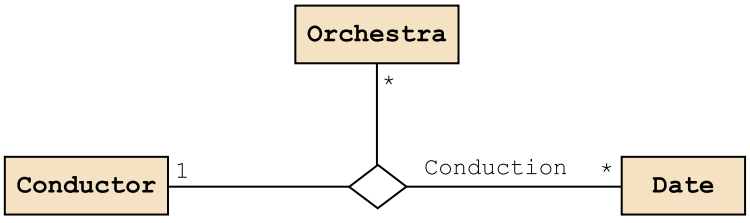


Este modelo es erróneo porque el compositor de una obra no depende de las orquestas que la interpretan. Es decir, hay una relación directa entre la obra y el compositor y otra entre la orquesta y la obra. El modelo correcto sería el que se muestra en la figura 8.

Figura 8. El esquema correcto

Esto no significa que no pueda haber algún tipo de relación de grado mayor que 2 y con alguna multiplicidad igual a 1. Como ejemplo correcto de tipo de relación ternaria en el que uno de los tipos de entidad participa con multiplicidad igual a 1, consideremos el de la figura 9, donde asumimos que en un momento dado una orquesta solo tiene un director, a pesar de que permitimos que un director se haga cargo de más de una orquesta de manera simultánea.

Figura 9. Un tipo de relación ternaria correcto con una multiplicidad igual a 1



2.5. Semántica de los tipos de entidad

La trampa conocida como *trampa de semántica del tipo de entidad* es la última trampa que presentamos y que deberemos evitar. Tal como indica su nombre, está relacionada con la interpretación que hagamos de la semántica de un tipo de entidad.

Decimos que caemos en la **trampa de semántica de tipo de entidad** cuando se asignan a un tipo de entidad atributos que no le corresponden debido a una interpretación incorrecta del significado del tipo de entidad.

Una manera de confirmar o descartar que un atributo se encuentra en un tipo de entidad que le corresponde es no perder de vista la semántica del tipo de entidad y comprobar si el atributo es coherente con esta semántica.

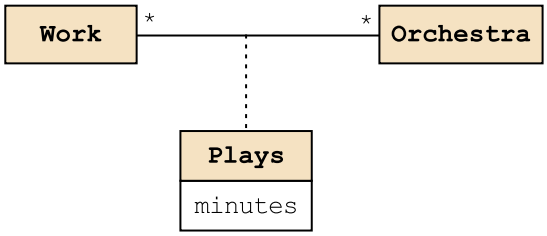
Supongamos el modelo conceptual que se presenta en la figura 10, en la cual definimos el atributo *minutes* en el tipo de entidad *Work*.

Figura 10. La trampa de la semántica de los tipos de entidad

Si reflexionamos un poco, no obstante, podremos concluir que los minutos de duración no corresponden tanto a la obra, sino a cada una de las interpretaciones que se hacen de ella. Por lo tanto, el atributo *minutes* tendría que ser un atributo del tipo de relación *Plays* y no del tipo de entidad *Work*. Desde el punto de vista semántico, fijémonos en que una obra es un concepto abstracto que hemos confundido con un concepto más físico, como es la interpretación de la obra.

El esquema conceptual correcto sería el que se presenta en la figura 11.

Figura 11. El esquema correcto



3. Diseño lógico: transformación del modelo conceptual en el modelo relacional

La etapa de diseño lógico consiste en obtener un esquema lógico a partir del esquema conceptual generado en la etapa anterior. El esquema lógico depende del tipo de base de datos elegido, aunque es independiente de la implementación concreta del sistema de gestión de bases de datos (SGBD[1]).

(1)SGBD es la sigla de *sistema de gestión de bases de datos*.

Como hemos mencionado, en este módulo nos centraremos en la conversión del esquema conceptual expresado en lenguaje UML en un esquema lógico para un tipo de base de datos relacional. En estos casos, el modelo lógico recibe el nombre de *modelo lógico relacional* o, simplemente, *modelo relacional*.

Reflexión

Dado que en este módulo trataremos únicamente el caso de modelos lógicos para bases de datos relacionales, cuando hablemos de *modelo lógico* interpretaremos que se trata de un modelo relacional.

Utilizaremos la palabra *relación* para referirnos al elemento básico del modelo relacional. Es preciso no confundirse con las relaciones a las que nos hemos referido al tratar el diseño conceptual, que eran instancias de tipos de relación. Hay que decir que el contexto nos tendría que ayudar a distinguir ambos conceptos de manera unívoca.

Actualmente, existen herramientas CASE[2] que son capaces de hacer este proceso de traducción de manera automática. Ahora bien, dado que hay situaciones en las cuales parte de un modelo se puede traducir de varias maneras, los usuarios de estas herramientas deben tener los conocimientos necesarios para realizar la traducción de forma manual porque, como usuarios de una herramienta CASE, deben ser capaces de modificar el resultado de la traducción automática si la herramienta no ha elegido la alternativa que se ajusta mejor a las condiciones de cada caso concreto. También es necesario tener estos conocimientos para decidir correctamente si la herramienta, al encontrarse ante alternativas que no puede resolver, pide la intervención del usuario.

(2)*CASE* es la sigla de la expresión inglesa *computer-aided software engineering*, 'ingeniería de software asistida por ordenador'.

Las herramientas *CASE*

Una herramienta *CASE* es la implementación de un conjunto de herramientas y de métodos para el desarrollo de software a partir de especificaciones y definiciones de alto nivel. Uno de los objetivos principales es liberar a los diseñadores de la base de datos de los procesos rutinarios, con posibilidad de automatización, no sólo para conseguir una producción más eficiente, sino también para evitar algunos de los errores que se pueden cometer en estos procesos de desarrollo. En el área de la modelización conceptual y las bases de datos, algunas de las herramientas más conocidas actualmente son VisualParadigm, ArgoUML, Poseidon o MagicDraw. Algunas herramientas *CASE* pueden efectuar ingeniería inversa, es decir, pueden ayudar a encontrar el esquema conceptual que corresponde a una base de datos ya diseñada e implementada.

A continuación, aunque se presupone que conocéis el modelo relacional y tenéis conocimientos básicos de SQL, mencionamos los conceptos más básicos de este modelo y su representación en lenguaje SQL para aquellos elementos que se deban considerar al transformar el modelo conceptual en el modelo lógico.

3.1. Conceptos previos del modelo relacional

El modelo relacional representa la información sobre la base de un conjunto de relaciones. Una **relación** se define como un conjunto de **atributos**, cada uno con un **dominio** concreto (el dominio es el conjunto de valores que se pueden asignar al atributo), y uno de estos es la clave primaria. Esta descripción se denomina **esquema de una relación**.

El conjunto de todos los esquemas de relación que describe la base de datos se denomina **esquema de la base de datos**.

Dado el esquema de la relación, podemos crear **tuplas** que conforman la extensión de la relación; cada tupla está informada con valores del dominio correspondientes a uno de sus atributos. Por ejemplo, podemos definir una relación *Work* cuyo esquema consta de dos atributos: un atributo *name*, cuyo dominio son las cadenas de caracteres, y un atributo *yearComp*, cuyo dominio es numérico. Por ejemplo, la extensión de la relación *Work* puede contener las tuplas <“Concierto para piano núm 3 de Mozart”, 1779> y <“Parsifal”, 1857>.

El modelo también permite expresar restricciones que limitan los valores o las combinaciones de valores que pueden tomar los atributos. Las más importantes son:

- **Clave candidata**. Un atributo o grupo de atributos constituye una clave candidata de la relación cuando no puede haber dos tuplas con el mismo valor en aquel atributo o grupo de atributos. Además, no es posible asignar valor nulo a estos atributos.

Reflexión

Una clave candidata sirve para identificar cada una de las tuplas de manera unívoca dentro de una relación.

Ejemplo

En una relación de orquestas en la que cada tupla contiene los datos de una orquesta, el atributo que corresponde al número de identificación fiscal se puede declarar como clave candidata.

- **Clave primaria**. De entre las claves candidatas, el diseñador elige una que será la que utilizaremos habitualmente para identificar de manera unívoca una tupla de la relación. En lenguaje SQL, la restricción *PRIMARY KEY* sirve para especificar la clave primaria de una relación.

- **Clave alternativa**. Cada una de las claves candidatas que no ha sido elegida clave primaria recibe el nombre de clave *alternativa*. En lenguaje SQL, se usa la cláusula *UNIQUE* se utiliza para especificar una clave alternativa en una relación.

Ejemplo

Siguiendo con el ejemplo anterior sobre la relación de orquestas, si añadimos un atributo con el nombre de orquesta, y éste es diferente para cada una, dicho nombre también puede ser clave candidata y, puesto que la otra es la primaria, esta sería una clave alternativa.

- **Clave foránea**. Se puede especificar que un atributo o conjunto de atributos de una relación *R1* forman una clave foránea, de una relación *R2* del esquema, cuando este atributo o conjunto de atributos permite referenciar alguna clave candidata de *R2*. Esta clave candidata de *R2* deberá estar formada por un conjunto de atributos que se corresponden uno a uno con los de la clave foránea de *R1*. La declaración de clave foránea implica que para cada tupla *t* de *R1* debe existir una tupla de *R2* que tenga como valor de los atributos de la clave candidata referenciada los mismos valores que tiene la tupla *t* en los atributos de la clave foránea. De manera alternativa, *t* puede tener valores nulos en los atributos de la clave foránea. En lenguaje SQL, la restricción *FOREIGN KEY* permite especificar la clave foránea de una relación.

  Ejemplo

  Si tenemos definidas las relaciones *Work* (con un atributo *workName* y un atributo *comp*) y *Composer* (con un atributo *compName* –que es clave primaria– y un atributo *yearBorn*) y decimos que el atributo *comp* de *Work* es clave foránea que referencia *Composer* por medio de *compName*, entonces para cada obra que no tenga el valor nulo en *comp* debe existir un compositor con este valor en el atributo *compName*.

- **Valores nulos**. Decimos que no admiten valores nulos todos aquellos atributos que siempre deben estar informados. En lenguaje SQL, un atributo de este tipo se especifica mediante la restricción *NOT NULL* aplicada a la columna en cuestión.

- **Comprobación de una condición**. Es una restricción que verifica que el valor de uno o más atributos satisface una expresión booleana que se especifica en la declaración de la restricción. En lenguaje SQL, usamos la restricción *CHECK* seguida de la expresión que debe satisfacerse.

  Ejemplo

  Si la relación de orquestas utilizada anteriormente tiene un atributo numérico denominado *numberMusicians* y queremos que todas las orquestas presentes en la relación tengan 30 músicos o más, estableceremos la restricción *Check(numberMusicians >= 30)*.

Por claridad y concisión, expresaremos un esquema del modelo relacional mediante una notación simplificada del lenguaje SQL estándar.

El estándar SQL

El estándar SQL es un lenguaje que permite trabajar con bases de datos relacionales. Es muy utilizado y casi todas las aplicaciones desarrolladas sobre bases de datos relacionales lo utilizan. Los SGBD relacionales que se eligen de manera habitual para almacenar la información lo implementan, a veces con pequeñas diferencias. Contiene sentencias tanto para definir bases de datos como para hacer actualizaciones y consultas. Ha tenido tres versiones principales que han aparecido, respectivamente, en los años 1989, 1992 y 1999. El lenguaje continúa evolucionando y se le incorporan nuevos elementos en sucesivas revisiones, la última publicada en el año 2011.

A continuación, se define brevemente la notación utilizada en estos materiales:

- Denotaremos las relaciones a partir del nombre, seguido de la lista de atributos entre paréntesis y separados por comas.
- Denotaremos las claves primarias subrayando con una línea continua los atributos que las forman.
- Denotaremos las claves alternativas subrayando con una línea discontinua los atributos que las forman.
- Denotaremos las claves foráneas mediante notación textual en la que indicaremos qué atributos son claves foráneas y a qué relaciones referencian.
- Utilizaremos el tipo de letra **negrita** en los nombres de atributo que queremos declarar *NOT NULL*.

Expresión de un modelo con dos relaciones

A continuación mostramos cómo expresar un modelo con dos relaciones, una de obras y una de compositores, en el que las obras se identifican mediante un código de catalogación y contienen un nombre popular que se puede repetir en compositores diferentes, pero no en obras de un mismo compositor. Los compositores se identifican por su nombre y es obligatorio asignar valor al atributo de fecha de nacimiento (*dateB*).

```
Composer (name, dateB, dateD)

Work (code, namePopular, author, date)
          {author} is foreign key to Composer
```

3.2. Impacto del uso de los valores nulos

Para empezar, presentaremos una breve reflexión sobre los valores nulos que inicialmente se incorporaron al modelo relacional para poder expresar el hecho de que un dato tiene un valor desconocido o es inaplicable.

Es importante conocer el impacto que puede tener la existencia de valores **nulos**, porque al llevar a cabo la traducción del esquema conceptual al esquema relacional, pueden surgir atributos que no aparecían en los tipos de entidad del esquema conceptual y que pueden admitir valores nulos.

Reflexión

Recordemos que los valores nulos constituyen un mecanismo que soluciona el problema de representación de datos desconocidos o que no se pueden aplicar. En general, no podemos evitar la existencia de valores nulos.

Un mal uso de los valores nulos puede causar problemas, básicamente de dos tipos: de eficiencia y de construcción de consultas que manipulan atributos que contienen valores nulos.

El problema de eficiencia se deriva del acceso a filas que contienen columnas con valores nulos, las cuales podrían haberse evitado con un diseño alternativo de la base de datos. El ejemplo siguiente lo ilustra:

```
Work (code, author, name, nTenors, nBasses,
      nContraltos, nSopranos, nBaritones)
```

```
SELECT * FROM Work WHERE nTenors >= 1
```

La relación *Work* nos dice cuántos tenores, bajos, contraltos, sopranos y barítonos se requieren para interpretar cada obra (las columnas con un nombre que empieza por *n* tienen esta información). Si consideramos que existirán obras sin intervención de voz, estas obras tendrán valores nulos en las columnas que empiezan por *n*, puesto que esta información no es aplicable a obras instrumentales. La consulta del ejemplo, que pretende localizar las obras en las que hay uno o más tenores, deberá acceder a todas las obras instrumentales, y lógicamente, no devolverá ninguna de estas tuplas como resultado de la consulta. Podemos diseñar un modelo relacional que evite el uso de valores nulos y también el acceso innecesario a tuplas. Por ejemplo, se puede hacer separando las obras en dos relaciones: una de obras con intervención de voces y otra de obras sin intervención de voces. Con esta idea, obtenemos el modelo relacional siguiente:

```
WorkInstr (code, author, name)
WorkChoral (code, author, name, nTenors, nBasses,
            nContraltos, nSopranos, nBaritones)
```

Con este modelo relacional, ahora haremos la consulta de este modo:

```
SELECT * FROM WorkChoral WHERE nTenors >= 1
```

La diferencia de accesos entre una alternativa y otra dependerá de la proporción de obras corales respecto al número total de obras. Cuantas menos obras corales (y, por lo tanto, más tuplas con valores nulos en la primera alternativa), más diferencia habrá.

Así pues, cuando hay columnas para las cuales una proporción grande de las filas puede tener valor nulo, hay que analizar si las consultas pueden ser penalizadas y, si procede, cambiar el diseño para eliminar la presencia de valores nulos.

En cuanto a la corrección de las consultas que involucran atributos que pueden tomar valor nulo, debemos estar atentos para asegurarnos de que la consulta devuelve el resultado correcto, tanto en el caso de existir tuplas con valor nulo como en el caso contrario. Consideremos el caso siguiente, en el que tanto la relación de directores como la de compositores tienen una columna que indica de qué país es el director o compositor. Si queremos obtener la de los directores que son de un país donde no ha nacido ningún compositor, podríamos pensar en las dos consultas que se muestran a continuación:

```
Conductor (name, country)
Composer (name, yearB, yearD, country)
```

```
SELECT * FROM Conductor
WHERE country NOT IN (SELECT country FROM Composer)

SELECT * FROM Conductor d
WHERE NOT EXISTS
(SELECT * FROM Composer c WHERE d.country = c.country)
```

Las dos consultas devuelven el mismo resultado siempre que la columna *country* no tenga valores nulos. Observad que si un director tiene el valor nulo en la columna *country*, la primera consulta no lo incorpora en el resultado y, en cambio, la segunda sí.

Este es solo un ejemplo para ilustrar el impacto que puede tener la existencia de valores nulos en las consultas. Será necesario, pues, tenerlo presente en todas las consultas que involucran los valores nulos en la condición de selección, atributos de agrupación (*GROUP BY*), funciones de agregación (*MAX*, *SUM*, ...), etc.

Después de los preliminares, pasemos a ver cómo podemos transformar cada uno de los elementos del modelo conceptual expresados en lenguaje UML en el modelo lógico relacional.

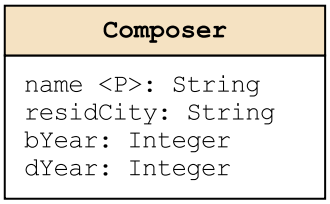
3.3. Tipo de entidad

El primer elemento del modelo conceptual que transformaremos será el concepto de *tipo de entidad*.

El **tipo de entidad** del esquema conceptual se transforma en general en una relación del modelo relacional. Cada **atributo** del tipo de entidad se convertirá en una columna de la relación.

Supongamos el tipo de entidad de la figura 12:

Figura 12. Un tipo de entidad



Vemos que este tipo de entidad tiene los atributos nombre (*name*), que es clave primaria, ciudad de residencia (*residCity*), que es una cadena de caracteres, año de nacimiento (*bYear*) y año de defunción (*dYear*), que son enteros. Este tipo de entidad se representa en el modelo lógico como la relación que se muestra a continuación:

Composer (name, **residCity, bYear, dYear**)

3.3.1. Atributos multivaluados

La transformación de los atributos multivaluados requiere un análisis más detallado. El modelo relacional no soporta directamente esta posibilidad, pero hay dos maneras de representar atributos multivaluados:

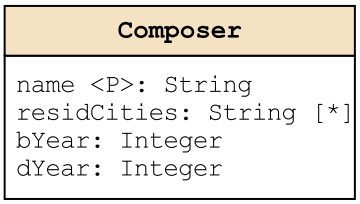
1) Por columnas. La representación por columnas consiste en definir en el esquema relacional tantas columnas como el número máximo de valores que pueda tomar el atributo multivaluado del esquema conceptual. Esta representación requiere conocer el número máximo de valores, información que no siempre está definida.

2) Por filas. Esta segunda representación consiste en representar cada valor de un atributo multivaluado como una fila o tupla de una nueva relación en el esquema relacional.

Ejemplo de atributo multivaluado

Supongamos, por ejemplo, que el tipo de entidad *Composer* ahora tiene, en vez del atributo *residCity*, un atributo *residCities* multivaluado, tal y como se muestra en la figura 13.

Figura 13. Un tipo de entidad con un atributo multivaluado



Suponiendo que cada compositor puede tener como máximo cinco ciudades de residencia, podemos representar este atributo multivaluado mediante dos modelos lógicos posibles:

1) Por columnas:

```
Composer (name, bYear, dYear, city1, city2,
          city3, city4, city5)
```

2) Por filas:

```
Composer (name, bYear, dYear)

Cities (comp, n, city)
       {comp} is foreign key to Composer
```

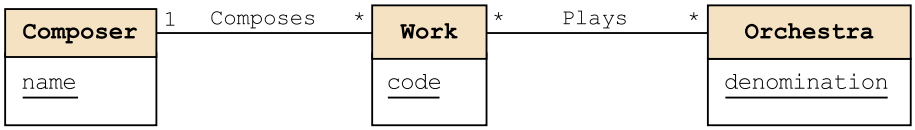
Fijémonos en que, como hemos explicado, para hacer la representación por columnas debemos conocer el número máximo de ciudades en las que puede haber vivido un compositor, y que para cada compositor dejaremos con valor nulo los atributos de ciudad que no sean necesarios. Si conocemos este número máximo y la mayoría de los compositores dejan pocos o ningún valor nulo, esta puede ser una buena representación. Si este no es el caso, probablemente será mejor la representación por filas. También hay que tener presente que en esta segunda representación será preciso efectuar operaciones de combinación (*join*) para recuperar las ciudades de un compositor, cosa que no habrá que hacer si se utiliza la primera representación.

3.4. Tipo de relación

Para realizar la transformación de los tipos de relaciones al modelo relacional deberemos fijarnos en la aridad, o grado, y en las multiplicidades, o conectividad. En lo que respecta al grado, distinguiremos entre grado 2 o más, y en cuanto a la conectividad, distinguiremos si el mínimo es 0 o más y si el máximo es 1 o más. También estudiaremos los casos particulares de tipos de relaciones reflexivas y de composiciones.

Todo **tipo de relación** se puede representar con una nueva relación, que tiene como clave primaria la concatenación de claves primarias de las relaciones que representan los tipos de entidad que participan en el tipo de relación. Además, esta nueva relación tendrá una clave foránea por cada tipo de entidad relacionado.

Figura 14. Un esquema con tipos de relaciones binarias



Ejemplo de tipos de relación

La relación "interpreta" (*Plays*) de la figura 14 se puede transformar en el modelo relacional de la manera siguiente:

```
Work (code)

Orchestra (denomination)

Plays (work, orchestra)
      {work} is foreign key to Work
      {orchestra} is foreign key to Orchestra
```

3.4.1. Tipos de relaciones binarias con una multiplicidad 1

Los **tipos de relaciones binarias 1..1 o 1..*** se pueden representar mediante una clave foránea en el extremo opuesto al que tiene la multiplicidad máxima igual a 1.

De este modo, el tipo de relación *composes* de la figura 14 se representa:

```
Composer (name)

Work (code, composer)
     {composer} is foreign key to Composer
```

Si además tenemos en cuenta que, tal y como especifica el esquema conceptual, toda obra tiene un compositor, entonces habrá que añadir la restricción *NOT NULL* sobre la columna *composer*. Por lo tanto, el modelo conceptual de la figura 14 se transforma en el siguiente modelo lógico:

```
Composer (name)

Work (code, composer)
     {composer} is foreign key to Composer

Orchestra (denomination)

Plays (work, orchestra)
      {work} is foreign key to Work
      {orchestra} is foreign key to Orchestra
```

Detengámonos y reflexionemos un poco sobre esta representación en el supuesto de que la multiplicidad mínima del tipo de relación *composes* fuera igual a 0 en lugar de igual a 1. Si suponemos que una obra puede no tener asociado ningún compositor –por ejemplo, porque se desconoce o consta como anónima–, entonces la clave foránea podría tomar valores nulos.

Si queremos evitar la existencia de valores nulos, podemos optar por utilizar la representación de los tipos de relación mediante una relación.

En nuestro ejemplo, si representamos *composes* por medio de una relación, obtendremos:

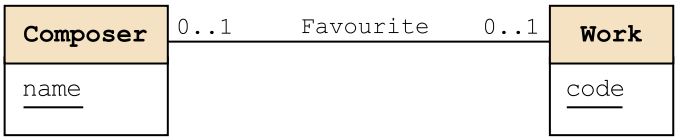
```
Composer (name)

Work (code)

Composes (work, composer)
         {work} is foreign key to Work
         {composer} is foreign key to Composer
```

En el caso particular de un tipo de relación binaria 1..1 entre dos tipos de entidad *E1* y *E2* que se transforman en las relaciones *R1* y *R2* del modelo lógico, podremos representar el tipo de relación como relación, como clave foránea de *R1* que referencia *R2* o como clave foránea de *R2* que referencia *R1*.

En caso de que las multiplicidades mínimas sean igual a 0, la representación por clave foránea tendrá que admitir valores nulos, tal y como hemos explicado. Si optamos por la representación por relación, hay que tener presente que ahora la relación no tiene una clave compuesta, sino dos claves candidatas.

Por ejemplo, si queremos representar el hecho de que un compositor puede tener una obra preferida (*Favourite*) y que una obra puede ser la preferida de un compositor, entonces tenemos el esquema conceptual de la figura 15.

Figura 15. Un tipo de relación con multiplicidades mínimas igual a 0



Si optamos por una transformación por relación, obtenemos el modelo lógico siguiente:

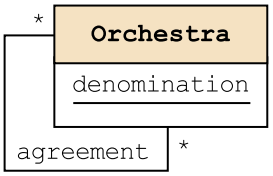
```
Composer (name)

Work (code)

Favourite (work, composer)
          {work} is foreign key to Work
          {composer} is foreign key to Composer
```

3.4.2. Tipos de relaciones binarias reflexivas

Otro caso particular que tenemos que estudiar es el de los tipos de relaciones binarias reflexivas. Se pueden representar igual que en el caso general: mediante claves foráneas o mediante relaciones.

Como ejemplo de transformación de tipo de relación reflexiva, consideremos el esquema de la figura 16, que modeliza los convenios que pueden existir entre orquestas.

Figura 16. Un tipo de relación reflexiva



El modelo relacional correspondiente es:

```
Orchestra (denomination)

Agreement (orc1, orc2)
          {orc1} is foreign key to Orchestra
          {orc2} is foreign key to Orchestra
```

En el supuesto de que el tipo de relación tenga la multiplicidad máxima mayor que 1, en ambos extremos se tiene que representar por relación, y en este caso hay que analizar si es simétrica.

Tipos de relaciones binarias

Un tipo de relación binaria es simétrico si para toda relación del tipo (*a* ,*b*) también existe la relación (*b*, *a).* Por ejemplo, el tipo de relación *hermanos* entre dos entidades de tipo *Persona* es simétrico.

En caso afirmativo, podemos optar por ahorrar espacio guardando la mitad de la información y dejando implícita la otra mitad, que se puede deducir por simetría. También hay que garantizar que para cada par (o_1, o_2) de la extensión exista el par (o_2, o_1):

- bien físicamente, con aserciones que lo garanticen,

Ved también

El mecanismo de aserciones se presenta en el subapartado 3.7 de este módulo.

- bien virtualmente, guardando solo la mitad de los pares y con aserciones que eviten la existencia de pares simétricos y una vista definida sobre la tabla. Esta vista debe reconstruir la extensión entera a partir de la mitad que tenemos almacenada.

Recordad

Una vista se declara dándole un nombre y definiéndola como una consulta sobre una o más tablas. Una vez se ha declarado, se le pueden hacer consultas como si fuera una tabla.

3.4.3. Tipos de relaciones binarias de composición

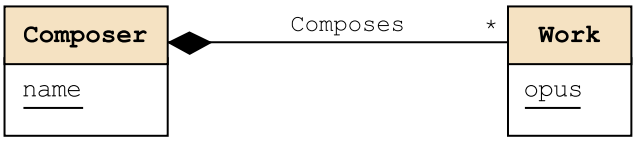
Las composiciones son, en el fondo, un caso especial de tipo de relación binaria y, además, se transforman en el modelo relacional del mismo modo. Por este motivo, las tratamos en este subapartado como otro caso de tipo de relación binaria.

Las composiciones establecen una relación de dependencia de existencia entre dos tipos de entidad. Normalmente, esta dependencia hace que la identificación del componente sea solo válida en el conjunto de componentes de un mismo compuesto.

Este **tipo de relación de composición** se representa como clave foránea en la relación que representa el tipo de entidad dependiente. Además, la clave primaria de este tipo de entidad está formada por el identificador del tipo de entidad concatenado con la clave foránea.

Imaginemos que queremos representar el conjunto de obras de cada compositor. Cada obra se identifica por el atributo *opus* (un número que ordena cronológicamente las obras de un compositor), pero diferentes obras de distintos compositores pueden tener un mismo opus. En la figura 17 se representa el esquema conceptual correspondiente.

Figura 17. Ejemplo de relación binaria de composición



De acuerdo con la regla de transformación de los tipos de relación de composición vista anteriormente, obtenemos el modelo lógico siguiente:

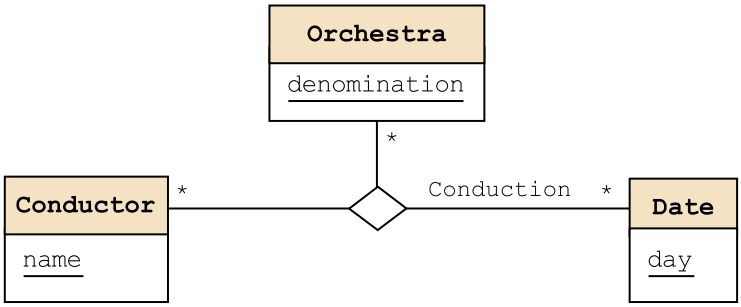
```
Composer (name)

Work (opus, comp)
     {comp} is foreign key to Composer
```

3.4.4. Tipos de relaciones *n*-arias

Los **tipos de relación de aridad mayor que 2** se representan por relaciones. Lo que es preciso tener en cuenta son las multiplicidades, puesto que si hay alguna conectividad máxima igual a 1, aparecen claves alternativas.

Supongamos que queremos registrar los datos históricos referentes a las direcciones de orquesta con el correspondiente director, tal y como se muestra en la figura 18.

Figura 18. Un tipo de relación ternaria



La figura 18 representa el caso sencillo, sin ninguna multiplicidad limitada a 1. El modelo relacional correspondiente es el siguiente:

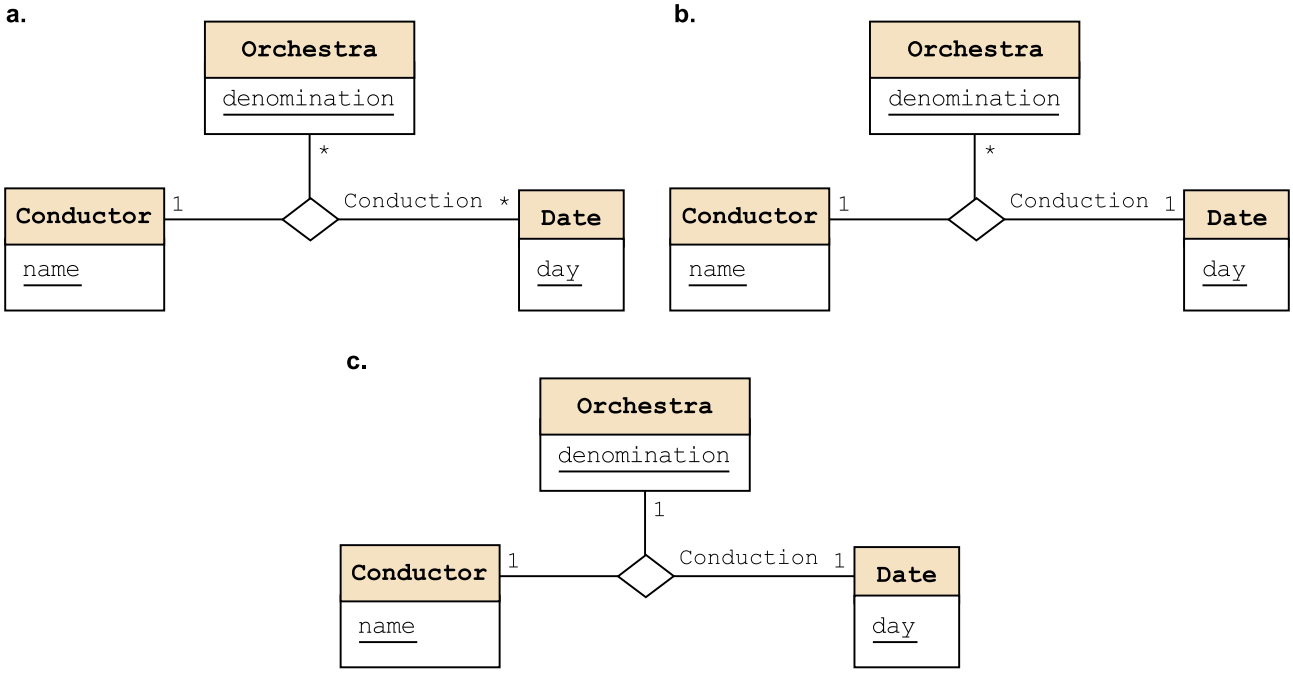
```
Orchestra (denomination)

Conductor (name)

Date (day)

Conduction (cond, orc, date)
           {cond} is foreign key to Conductor
           {orc} is foreign key to Orchestra
           {date} is foreign key to Date
```

El hecho de que el extremo de un tipo de entidad participante tenga conectividad máxima igual a 1 indica que, una vez fijadas las otras entidades, queda ya determinada la primera. Por lo tanto, el atributo correspondiente a esta entidad no forma parte de la clave de la relación en la que se transforma el tipo de relación *n*-aria. Si hay más de un participante con multiplicidad máxima igual a 1, podemos formar varias claves según qué atributo sea descartado para formar la clave.

En la figura 19 podemos ver diferentes casos en el mismo contexto, en los que vamos limitando la multiplicidad máxima de los extremos del tipo de relación *Conduction*.

Observemos el impacto que tiene en la relación *Conduction* y cómo afecta la definición de las claves candidatas.

Figura 19. Tipos de relaciones ternarias con multiplicidades mínimas igual a 1



Considerando los esquemas de la figura 19, obtenemos:

a) una clave candidata, pero formada solo por dos atributos. El otro se tiene que declarar *NOT NULL*:

```
Conduction(cond, orc, date)
```

b) dos claves candidatas:

```
Conduction(cond, orc, date)
           ------------
```

c) tres claves candidatas:

```
Conduction(cond, orc, date)
           - - - - - -
           -------  -------
```

De manera similar, se representan los tipos de relación de aridad mayor que 3, a pesar de que son poco frecuentes.

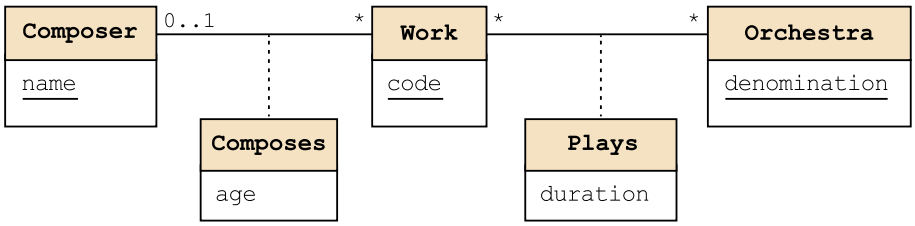
3.5. Tipos de entidades asociativas

Las entidades asociativas del modelo relacional también deben transformarse en el modelo lógico, y por este motivo debe tenerse en cuenta dónde se ubican los atributos de este tipo de entidad, puesto que conceptualmente pertenecen al tipo de relación.

En **un tipo de entidad asociativa**, los atributos se pueden representar de dos maneras, dependiendo de cómo se ha representado el tipo de relación, si como una relación o como una clave foránea. En el primer caso, los atributos del tipo de relación se convertirán en columnas de la relación y, en el segundo caso, en columnas de la clave foránea de la relación.

Veámoslo con un ejemplo. Supongamos el modelo conceptual de la figura 20.

Figura 20. Tipos de entidades asociativas



Dado que *Plays* tiene una multiplicidad ***..***, se transforma mediante una relación. En cambio, puesto que *composes* tiene una multiplicidad 0..1, se puede transformar mediante una clave foránea. Teniendo en cuenta esto, la traducción al modelo relacional del esquema conceptual es la siguiente:

```
Orchestra (denomination)

Composer (name)

Work (code, comp, age)
     {comp} is foreign key to Composer

Plays (work, orchestra, duration)
      {work} is foreign key to Work
      {orchestra} is foreign key to Orchestra
```

Tenemos que prestar atención una vez más a los valores nulos. El atributo *comp* puede ser nulo porque el tipo de relación tiene multiplicidad 0..1 y, por lo tanto, el atributo *age* también lo puede ser.

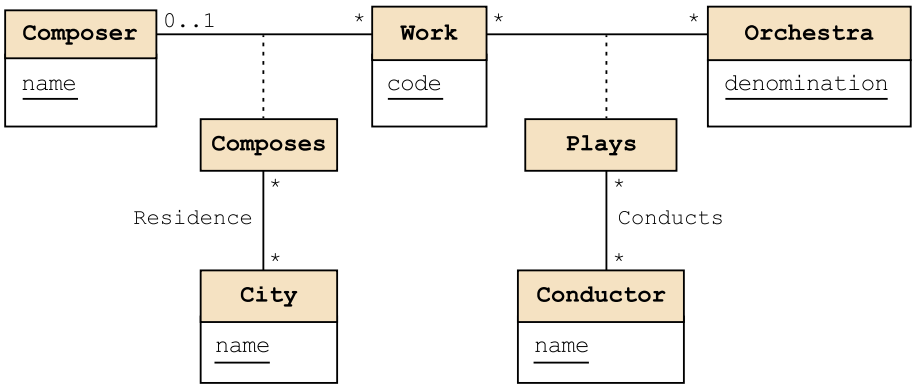
Es preciso tener en cuenta que si las claves foráneas que representan un tipo de relación pueden ser nulas, todos los atributos que tenga el tipo de relación también lo pueden ser.

Un tipo de entidad asociativa puede participar en tipos de relación. Al igual que el resto de los tipos de entidades participantes de un tipo de relación, también los tipos de entidad asociativa tendrán que ser referenciados por alguna clave foránea si participan en tipos de relaciones.

Si el tipo de entidad asociativa se ha representado con una relación, la clave primaria será compuesta y las claves foráneas que la referencien también lo deberán ser. En cambio, si se ha representado como clave foránea en una relación *R*, las claves foráneas que tengan que referenciar el tipo de entidad asociativa deberán referenciar la relación *R*.

Para ilustrar esta casuística, fijémonos en el modelo conceptual de la figura 21. Este modelo relaciona cada obra con su compositor y las orquestas que la han interpretado. Además, se incluye en qué ciudades residía el compositor durante la composición de la obra y qué directores dirigían la orquesta en las interpretaciones de la obra.

Figura 21. Ejemplo de tipos de entidades asociativas que participan en otros tipos de relaciones



De acuerdo con la regla de transformación de los tipos de entidad asociativa comentada anteriormente, obtenemos el modelo relacional siguiente:

```
City (name)

Composer (name)

Orchestra (denomination)

Conductor (name)

Work (code, comp)
     {comp} is foreign key to Composer

Residence (city, cmp)
          {city} is foreign key to City
          {cmp} is foreign key to Work

Plays (work, orchestra)
      {work} is foreign key to Work
      {orchestra} is foreign key to Orchestra

Conducts (wo, or, cond)
         {wo, or} is foreign key to Plays
         {cond} is foreign key to Conductor
```

Observemos que el tipo de entidad asociativa *Plays* se ha transformado en la relación *Plays*. Por este motivo, en el esquema relacional, *Conducts* referencia *Plays*. En cambio, el tipo de entidad asociativa *Composes* se ha transformado en la clave foránea *comp* de la relación *Work*. Por este motivo, *Residence* referencia *Work*, que es donde ha quedado la clave foránea *comp*, la transformación de *Composes*.

3.6. Generalizaciones

La última estructura del modelo conceptual de la cual veremos su tranformación en el modelo relacional es la generalización/especialización o, simplemente, generalización.

Recordad

Toda generalización/especialización presenta una superclase (o tipo de entidad genérica) y un conjunto de subclases (o tipos de entidades específicas). Además, es posible que estas subclases sean disjuntas o solapadas, por un lado, y que la especialización sea total o parcial, por el otro.

Hay tres maneras de transformar una generalización en el modelo relacional:

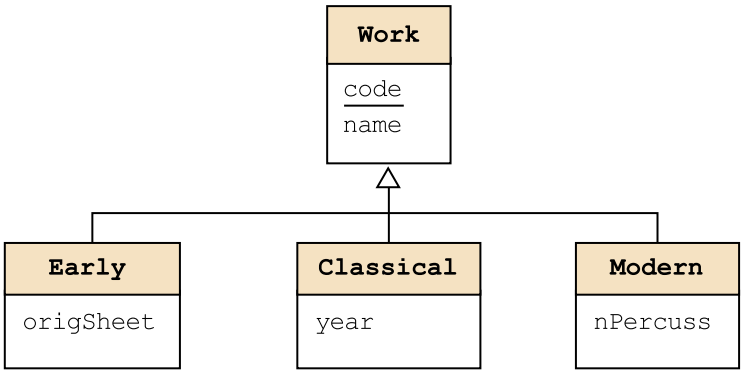
1) Una única relación, cuyas columnas son la unión de los atributos de todos los tipos de entidad (superclase y subclases).

2) Una relación para cada subclase, pero ninguna relación para la superclase. Cada relación tendrá las columnas que corresponden a los atributos de su subclase y, además, los de la superclase.

3) Una relación para cada tipo de entidad. Cada relación contendrá las columnas de los atributos correspondientes a su tipo de entidad. En el caso de las relaciones que representan las subclases, además, tendremos el identificador con una restricción de clave foránea que referenciará la relación padre o superclase.

Consideremos el esquema conceptual de la figura 22, que representa diferentes tipos de obra. Están las obras de música antigua (*early*) de las cuales queremos información sobre la partitura original (*original sheet*), las de música clásica (*classical*) y las de música moderna (*modern*).

Figura 22. Un caso de generalización/especialización



A continuación, veamos cómo se transforma según las diferentes opciones:

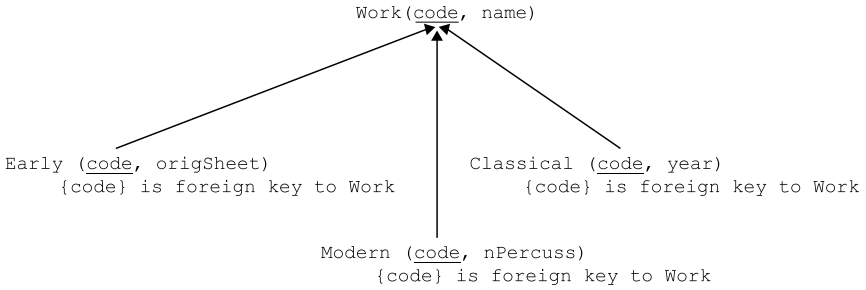
1.

```
Work(code, name, origSheet, year, nPercuss)
```

2.

```
Early(code, name, origSheet)
Classical(code, name, year)
Modern(code, name, nPercuss)
```

3.



Para elegir entre las tres representaciones posibles de una generalización, tenemos que considerar una serie de circunstancias:

1. Una primera cuestión que hay que tener en cuenta es cuáles de las posibles estructuras pueden almacenar toda la información de todas las posibles instancias.

2. Otro criterio importante es el número de valores nulos generados en cada estructura.

3. También hay que decir que algunas de las opciones pueden implicar redundancia cuando la generalización es solapada. Por ejemplo, en la segunda opción, una instancia que pertenezca a dos subclases repetirá dos veces los valores de los atributos provenientes de la superclase.

d) Finalmente, también en función de la consulta que queramos realizar, debemos considerar cuestiones de rendimiento. En el ejemplo anterior, cuando se intenta combinar las relaciones más específicas con la relación genérica o padre, la opción 3 genera operaciones de combinación (*join*) entre las relaciones para obtener todos los resultados, mientras que la opción 2 genera operaciones de unión (*union*) entre las diferentes relaciones para acceder a todos los resultados.

Comentemos algunos ejemplos para ilustrar estos criterios:

a) Si elegimos la opción 2 y se trata de una generalización parcial, no dispondremos de lugar donde almacenar las instancias que no pertenecen a ninguna subclase.

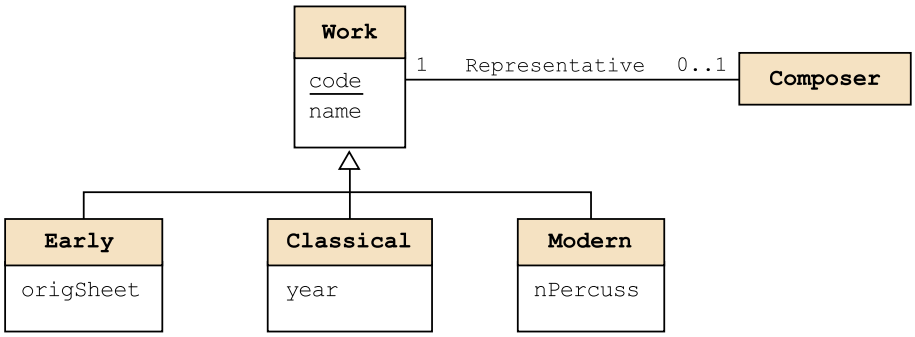
b) Si elegimos la opción 1, cada tupla tendrá valores nulos en las columnas que no corresponden a la subclase de la instancia que se considere. Si la generalización es total y solapada, toda instancia será de una o más subclases y la proporción de nulos quizás será admisible. En el otro extremo, una generalización parcial y disjunta dará lugar a una gran proporción de nulos.

c) Si elegimos la opción 3, en el caso de una generalización solapada, los atributos comunes de la superclase se repetirán para cada instancia de la relación tantas veces como el número de subclases a las que pertenece la instancia.

Hay que elegir la alternativa que permita almacenar todas las instancias y que no genere valores nulos (o que genere un número mínimo de estos) ni redundancia, excepto cuando haya razones de rendimiento que exijan una opción diferente.

A continuación, podemos ver otro ejemplo en el que la superclase forma parte de un tipo de relación. La figura 23 muestra un modelo en el cual se define que cada compositor tiene una obra representativa y que las obras pueden ser de diferentes tipos.

Figura 23. Un caso de generalización/especialización en el que la superclase participa en un tipo de relación



La regla de transformación de generalizaciones se debe considerar como una guía general, pero además hay que tener en cuenta el resto del esquema conceptual en el que participa la generalización. Por ejemplo, si hay referencias de otras relaciones (surgidas de la traducción de otra parte del esquema), algunas de las opciones dejan de ser válidas: en el esquema de la figura 23, si decidimos convertir el tipo de relación *Representative* en una clave foránea que referencia *Work*, la opción 2 no es válida porque necesitamos una relación para la clase genérica.

3.7. Restricciones

Finalmente, nos falta analizar la manera en que las restricciones que forman parte del modelo conceptual se transforman en el modelo lógico relacional.

Es importante que las **restricciones** del modelo conceptual se conserven en el modelo lógico. Algunas de estas restricciones son estructurales del diagrama de clases, y otras son restricciones textuales que el diseñador ha indicado en el modelo conceptual.

En este subapartado, nos centraremos en las restricciones estructurales del diagrama de clases. Algunas restricciones quedan incorporadas en el modelo relacional si se han aplicado de manera correcta las transformaciones que hemos visto hasta ahora. Hay, no obstante, otras restricciones que habrá que añadir al modelo lógico relacional de manera más explícita.

Las restricciones estructurales son de tres tipos:

1) Restricciones de identidad. Corresponden a identificadores de los tipos de entidad. Ya hemos comentado que se incorporan como claves primarias o alternativas en el diseño lógico.

2) Multiplicidad de los tipos de relación. Hay que garantizar la conectividad entre las instancias de relaciones, respetando los máximos y mínimos de estas multiplicidades. Ya hemos tenido en cuenta algunas de estas restricciones, pero quedan otras que aún no hemos tratado.

3) Tipología de las generalizaciones. Se tiene que procurar que queden reflejadas las restricciones de pertenencia a superclases y a las subclases teniendo en cuenta su tipología (declaración de disjunto/solapado y total/parcial).

Los mecanismos de los que disponemos para mantener y controlar las restricciones son los siguientes:

- Restricciones admitidas en la creación de tablas: *PRIMARY KEY*, *FOREIGN KEY*, *UNIQUE*, *NOT NULL*, *CHECK*. Son mecanismos automáticos y de baja complejidad que resultan fáciles de definir y de mantener.

- Aserciones. El estándar SQL incorpora este mecanismo de definición de restricciones basado en condiciones que se especifican usando una construcción que puede involucrar instrucciones *SELECT*. Por este motivo, son muy potentes y permiten expresar condiciones que no pueden incluir las restricciones del subapartado anterior, las cuales sólo pueden acceder a información de una tupla de una relación. Se trata de un mecanismo con las características positivas de los anteriores (son automáticas, de baja complejidad, fáciles de definir y de mantener) pero que desgraciadamente ningún SGBD implementa hoy día.

- Procedimientos almacenados. Son procedimientos que pueden contener sentencias SQL (consulta, actualización y definición) combinadas con estructuras de control clásicas de programación (iteraciones, alternativas, etc.). Se pueden definir unos procedimientos de acceso a las tablas que efectuarán los controles necesarios para asegurar el mantenimiento de restricciones.

- Disparadores (en inglés, *triggers*). Son similares a procedimientos almacenados que se asocian a operaciones sobre tablas y que se ejecutan de manera automática a partir de determinadas acciones sobre los datos (inserción, modificación o eliminación de datos). Se trata de un mecanismo automático, pero requiere un esfuerzo considerable definirlos y mantenerlos. Podemos, pues, diseñar disparadores que se ejecuten cuando se realicen actualizaciones en los datos de tablas en las cuales haya que comprobar que no se violan las restricciones.

- Precondiciones. Se puede considerar la posibilidad de delegar el control de algunas restricciones a las aplicaciones y liberar la base de datos, la cual actuará bajo el supuesto de que las operaciones que se le piden satisfacen las condiciones de corrección necesarias.

- Control externo. Incluso se puede llevar la idea anterior más allá y confiar en que determinadas restricciones se controlen de manera externa a la aplicación, habitualmente por parte de algún mecanismo o proceso automático que valida los datos.

En este módulo ya hemos tratado las restricciones de identidad. A continuación, veremos cómo podemos tratar los otros dos tipos de restricciones estructurales y los problemas con los que nos podemos encontrar.

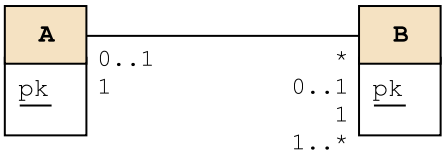
3.7.1. Multiplicidades

En lo que respecta a las multiplicidades, será necesario analizar los diferentes tipos de relación. Empecemos por los tipos de relaciones binarias que se representan mediante alguna clave foránea, es decir, tipos de relaciones entre dos relaciones *A* y *B* de manera que una de estas tiene una o más columnas que referencian las tuplas de la otra relación.

```
A (pk, …)

B (pk, fk, …)
  {fk} is foreign key to A
```

En la figura 24 se muestran las posibles multiplicidades de un tipo de relación binaria.

Figura 24. Conjunto de posibles multiplicidades de un tipo de relación binaria que se puede transformar en clave foránea



Según la multiplicidad de la izquierda (lado de la entidad *A*), la clave foránea admitirá valores nulos o no, como ya hemos discutido. La multiplicidad de la derecha (lado de la entidad *B*), sin embargo, nos puede obligar a añadir alguna restricción que antes no hayamos tenido en cuenta. Según el valor de esta multiplicidad, hay que considerar:

Ved también

Podéis ver las multiplicidades y los diferentes tipos de relación en el subapartado 3.4 de este módulo didáctico.

- Si es *, no hay que añadir nada más.

- Si es 0..1, tenemos que declarar que *fk* es *UNIQUE*, asegurando así que no puede haber más de un elemento de *B* asociado con el mismo elemento de *A*.

- Si es 1, tenemos que asegurar que toda tupla de *A* es referenciada por una tupla de *B*. Podemos expresar de varias maneras que toda fila de *A* es referenciada por una fila de *B*. Por ejemplo, como una aserción o una clave foránea de *A* hacia *B*. Esta clave foránea sería el atributo *pk* de *A* que haría referencia a *fk* de *B*, que habremos definido como *UNIQUE*. Además, será necesario controlar que si la tupla *b* de *B* hace referencia a la tupla *a* de *A*, la tupla *a* hace referencia a la tupla *b*. Esta comprobación se podría hacer, por ejemplo, empleando un disparador.

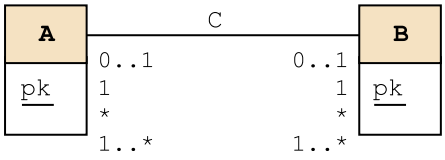
- Si es 1..*, tenemos que asegurar mediante una aserción que toda fila de *A* es referenciada por una o más filas de *B*.

A continuación, analizamos otro tipo de relación binaria.

```
A (pk, …)

B (pk, …)

C (fk1, fk2)
  {fk1} is foreign key to A
  {fk2} is foreign key to B
```

Las multiplicidades posibles de este tipo de relación son las que se muestran en la figura 25.

Figura 25. Posibles multiplicidades de un tipo de relación binaria que se puede transformar en relación



En función del número máximo de las multiplicidades de los tipos de entidades *A* y *B* en el tipo de relación *C* (si es igual a 1 o mayor que 1), *fk1* y *fk2* serán claves por sí mismas o no:

- si ambos máximos son iguales a 1, tendremos dos claves alternativas.
- si un máximo es igual a 1 pero el otro es mayor que 1, habrá una sola clave formada por una sola columna.
- si ambos máximos son mayores que 1, habrá una clave formada por las dos columnas.

Pensemos ahora en las multiplicidades mínimas, que se tratarán cada una de manera independiente: si el mínimo es igual a 0, no hay que añadir más controles. Si la multiplicidad a la izquierda de la figura 25 es igual a 1, tenemos que asegurar que todas las tuplas de la relación *B* aparezcan una y solo una vez en la relación *C*. Y si la multiplicidad es 1..* tendremos que asegurar que aparezcan una o más veces. Será preciso establecer los controles simétricos si el mínimo de la derecha de la figura 25 es igual a 1.

Reflexión

Encontramos situaciones análogas a las que hemos tratado en el caso de representación por clave foránea y que también solucionamos de manera análoga.

Para acabar el estudio de restricciones provenientes de tipos de relación, veremos ahora los tipos de relación de aridad superior a 2. Como ya hemos explicitado, en función del máximo de las multiplicidades (si es igual a 1 o mayor que 1), la clave de la relación del tipo de relación estará formada por todas las claves foráneas o solo por una parte de las mismas. También puede pasar, como ya hemos visto, que aparezcan claves alternativas.

Ved también

Podéis ver los tipos de relaciones *n*-arias en el subapartado 3.4.4 de este módulo didáctico.

En cuanto a las **multiplicidades mínimas**, en el caso de aridad mayor que 2 es habitual que sean 0. Si se da el caso de multiplicidad mayor que 0, será necesaria una aserción que compruebe que se dan efectivamente las relaciones indicadas por la multiplicidad.

Por ejemplo, en la transformación del esquema conceptual de la figura 19a, la aserción tendrá que comprobar que para todo par de orquesta y fecha hay un director relacionado.

3.7.2. Generalizaciones

Ved también

Podéis ver las generalizaciones en el subapartado 3.6 de este módulo didáctico.

En este subapartado nos ocuparemos de las restricciones correspondientes a las características de las generalizaciones. Según cuál sea la opción elegida de las tres presentadas, la que se utilice para transformar la generalización del modelo conceptual al modelo lógico, deberemos incorporar un tipo de restricciones u otro:

a) Una única relación para todos los tipos de entidad de la jerarquía. Es útil en casos muy particulares de generalizaciones solapadas y parciales, como ya se ha comentado. En función de cómo se indique a qué clases o subclases puede pertenecer una instancia en el modelo conceptual, será necesario diseñar alguna aserción para comprobar que cada instancia pertenece a alguna subclase.

b) Una relación para cada tipo de entidad específica. Se ajusta bien a los casos en que la jerarquía es disjunta y total. En este caso, será necesario controlar que una misma instancia no pertenezca a más de una subclase. Esta comprobación se puede realizar mediante el uso de disparadores incorporados en la inserción y la modificación.

c) Una relación para la entidad genérica. Es la opción más flexible y permite representar cualquier combinación disjunta/solapada y total/parcial. En este caso será necesario añadir los controles de disjunta (como acabamos de explicar) y/o total (que podemos implementar con una aserción) según proceda.

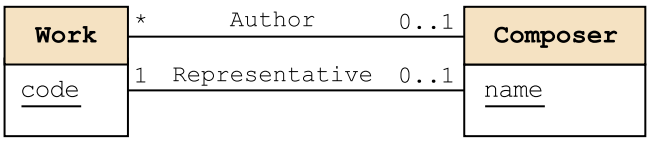
3.7.3. Abrazos mortales

Finalizaremos este subapartado de restricciones comentando un problema que puede aparecer en el proceso de incorporación de las restricciones en el modelo relacional, o también durante el paso del esquema conceptual al modelo relacional en determinadas situaciones. Nos referimos a los abrazos mortales, referencias cíclicas entre relaciones del modelo.

Cuando en un esquema relacional se genera un ciclo de claves foráneas, decimos que se ha producido un **abrazo mortal de definición**. Esto significa que no podemos definir directamente las tablas correspondientes porque cada una necesita que se haya definido previamente alguna otra. La solución de este problema pasa por definir primero una tabla sin clave foránea mediante una sentencia *CREATE TABLE*. A continuación, se definen otras tablas que requerían la primera, y finalmente se incorpora la clave foránea en la primera tabla con una sentencia *ALTER TABLE*.

Supongamos que queremos modelizar una situación en la cual una obra sea representativa de un compositor y que un compositor tenga o no una obra de referencia. El modelo conceptual para esta situación sería el que se describe en la figura 26.

Figura 26. Un tipo de relación binaria que se puede transformar en dos claves foráneas cíclicas



Este modelo conceptual se puede transformar en el modelo relacional siguiente:

```
Work (code, repr)
     {repr} is foreign key to Composer

Early (name)
      {name} is foreign key to Work
```

La clave foránea de *Work* en *Composer* representa el tipo de relación *Representative*. Le añadimos la clave foránea de *Composer* a *Work* junto con la restricción *UNIQUE* para garantizar la multiplicidad igual a 1. En un primer momento, nos encontramos con que no podemos definir estas relaciones porque ambas necesitan que la otra ya esté definida previamente. Para solucionarlo, crearemos primero una de las tablas sin clave foránea, después la otra tabla, y finalmente incorporaremos la clave foránea a la primera tabla.

Una vez superada la dificultad de la creación de tablas con referencias cíclicas, nos podemos encontrar con una situación similar a la hora de insertar filas en las tablas vacías: no podemos insertar una obra si no está antes el compositor, pero tampoco podemos insertar el compositor hasta que no tengamos su obra representativa.

Cuando en un esquema relacional no podemos insertar las tuplas que darían lugar a un contenido que no viola ninguna restricción porque hay un ciclo de claves foráneas que impide insertar las tuplas de una en una sin violar la integridad referencial, decimos que se ha producido un **abrazo mortal de carga**.

Una primera solución del abrazo mortal de carga consiste en eliminar las restricciones que la provocan o delegarlas a algún otro nivel. Podemos, sin embargo, mantener el control de estas restricciones en el ámbito del SGBD difiriendo su comprobación.

Los SGBD, por medio del mecanismo de transacciones, nos ofrecen la posibilidad de diferir la comprobación de restricciones en lugar de hacer la comprobación inmediatamente después de cada operación elemental. De este modo, después del primer *INSERT* se estará violando la restricción, pero el segundo *INSERT* nos llevará a un nuevo estado en el cual se satisface la restricción. En medio de las dos sentencias de inserción de filas, la SGBD impedirá los accesos a estas tablas que son, provisionalmente, inconsistentes. Una vez completadas las dos sentencias de inserción, el SGBD comprobará las restricciones y volverá a permitir el acceso a las tablas.

3.8. Reconsideraciones

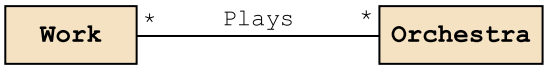
Una vez finalizada la transformación de un esquema conceptual en un modelo relacional, quedan detalles que debemos refinar en el esquema relacional obtenido.

En este subapartado, presentamos dos situaciones que pueden ayudar a ajustar el esquema lógico.

La primera de estas situaciones se presenta cuando existe la posibilidad de eliminar alguna relación que puede parecer innecesaria. Cuando nos encontramos con relaciones que consisten exclusivamente en la clave primaria y que además son referenciadas desde otra relación, nos podemos preguntar si es mejor eliminar estas relaciones y quedarnos sólo con la relación que las referencia, de manera que se evite repetir los mismos valores tanto en la clave foránea referenciadora como en la clave primaria referenciada. La respuesta suele ser negativa porque la existencia de estas relaciones nos permite comprobar la integridad referencial.

Consideremos el caso de la figura 27, que representa obras, orquestas y qué obras ha interpretado cada orquesta.

Figura 27. Un tipo de relación binaria



A partir del modelo conceptual de la figura 27, obtenemos el esquema lógico siguiente:

```
Orchestra (name)

Work (denomination)

Plays (work, orchestra, year)
      {work} is foreign key to Work
      {orchestra} is foreign key to Orchestra
```

Si *Work* y *Orchestra* no tienen más atributos que el identificador, nos podemos preguntar si es mejor eliminar estas relaciones y quedarnos sólo con la relación *Plays*, en la que ya aparecen estos identificadores. Debemos valorar, no obstante, que *Orchestra* y *Work* nos permiten estar seguros de que siempre que aparece una obra o una orquesta se trata de una obra u orquesta que existe y que siempre lo hace con el mismo nombre o denominación.

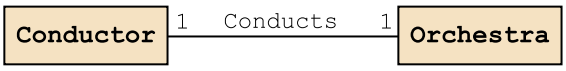
Si nos encontramos en una situación (que es poco frecuente) en la cual la integridad referencial nos viene garantizada por algún otro sistema (por ejemplo, no es necesaria una tabla de fechas porque la SGBD ya no permite la existencia de valores de fecha incorrectos), podremos eliminar la tabla.

Otra opción de simplificación consiste en fusionar dos tipos de entidad conectados por un tipo de relación 1..1 en un único tipo de entidad. Tomaremos la decisión sobre si es mejor fusionar o no en función del espacio ocupado y del tiempo requerido para las operaciones más frecuentes (si es más frecuente consultar atributos de un solo tipo de entidad o, por el contrario, consultar información que incluye atributos de los dos tipos de entidades). La fusión consiste en representar el esquema conceptual, no con dos relaciones vinculadas con claves foráneas, sino con una única relación que incluya los atributos de los dos tipos de entidad.

Si nos decidimos a llevar a cabo la fusión, la relación resultante tendrá dos claves alternativas y deberemos elegir cuál debe ser la primaria. El criterio de elección más adecuado suele ser el de mínima frecuencia de cambio.

Fijémonos en el caso de la figura 28, en el que se representan orquestas, directores y el director que dirige cada orquesta.

Figura 28. Dos tipos de entidad susceptibles de ser transformados en una única relación



Si fusionamos ambos tipos de entidad, la relación resultante tendrá dos claves alternativas (la de *Conductor* y la de *Orchestra*). En este caso, seguramente es mejor que la clave primaria sea el identificador de orquesta, que prácticamente no cambia nunca; en cambio, el director de una orquesta puede cambiar con mayor frecuencia.

4. Normalización

Durante el proceso de diseño se han tomado diferentes decisiones que determinan un diseño concreto, entre las distintas alternativas posibles, para resolver una misma necesidad. Por ejemplo, un mismo concepto del mundo real puede dar lugar a esquemas conceptuales ligeramente diferentes, o la transformación del modelo conceptual en el modelo lógico se puede llevar a cabo con diferentes alternativas o matices.

En los apartados anteriores, hemos visto algunos criterios para elegir o descartar determinadas opciones (por ejemplo, favorecer determinadas operaciones por encima de otras o evitar la existencia de valores nulos). En este apartado veremos las condiciones de normalización, que son las condiciones que garantizan que la base de datos está diseñada de manera que no se mezclen conceptos diferentes en una misma relación. Esta característica es positiva porque facilita la comprensión del diseño y evita redundancias innecesarias.

En primer lugar, veremos las anomalías que se pueden producir cuando una base de datos no está normalizada. Estas anomalías implican ineficiencia y complejidad en el mantenimiento de la coherencia de los datos. En segundo lugar, después de un repaso previo de conceptos del modelo relacional y del álgebra de conjuntos, presentaremos la teoría de normalización y veremos cómo la podemos aplicar a los esquemas lógicos. Veremos que esta aplicación elimina las anomalías de la base de datos.

4.1. Anomalías

Un diseño lógico no normalizado tiene como consecuencias negativas la falta de separación de conceptos y la existencia de redundancias que provocan la aparición de las denominadas **anomalías de actualización**. Decimos que hay anomalías cuando es preciso actualizar muchas tuplas para reflejar un cambio elemental que con un diseño normalizado implicaría un volumen de tuplas mucho menor.

Estas anomalías pueden aparecer en la inserción, la supresión o la modificación de tuplas.

Anomalías de inserción

Cuando resulta imposible insertar informaciones elementales de manera independiente en una base de datos, decimos que se produce una anomalía de inserción.

Ejemplo

Imaginemos que queremos almacenar información de las obras que tenemos disponibles en nuestra discoteca y de los compositores que son sus autores. Con este objetivo, creamos la relación siguiente:

`Compositions(work, composer, yearComp, bCentury, digitDegree)`

Se registra el año de composición de las obras (en el atributo *yearComp*), el siglo en que nacieron los compositores (*bCentury*) y el porcentaje de digitalización que han alcanzado sus obras (*digitDegree*). En un momento determinado, la relación podría tener la extensión que representa la figura 29.

Figura 29. Extensión de una relación con redundancias y anomalías

Compositions				
work	composer	yearComp	bCentury	digitDegree
Symphony 9	Mahler	1923	19	70
Symphony 5	Mahler	1918	19	70
Parsifal	Wagner	1857	19	42
Conc Piano 3	Mozart	1779	18	42

Si queremos añadir un nuevo compositor denominado *Montsalvatge* cuyo nombre y siglo de nacimiento conocemos pero de quien aún no disponemos de información de ninguna de sus obras, no podemos añadirlo. Tendríamos que insertar la tupla:

NULL	Montsalvatge	NULL	20	NULL

Sin embargo, no podemos porque el atributo *Work* es la clave primaria y ésta no puede tener valor nulo. Nos encontramos, pues, con que no podemos insertar información de un compositor independientemente de sus obras.

Anomalías de supresión

Cuando se produce una pérdida de información involuntaria de un hecho elemental debido a la supresión de otro hecho elemental, decimos que se produce una anomalía de supresión.

Ejemplo

Supongamos ahora que queremos suprimir el *Tercer concierto para piano* de Mozart. Después de la supresión, nuestra relación tendrá la extensión que muestra la figura 30.

Figura 30. Extensión de la relación después de una supresión

Compositions				
work	composer	yearComp	bCentury	digitDegree
Symphony 9	Mahler	1923	19	70
Symphony 5	Mahler	1918	19	70
Parsifal	Wagner	1857	19	42

Observad que, de manera involuntaria, hemos perdido la información que teníamos del compositor Mozart.

Anomalías de modificación

Cuando se presenta la necesidad de modificar varias (potencialmente muchas) tuplas para reflejar el cambio de un solo hecho elemental, se dice que se produce una anomalía de modificación.

Ejemplo

Si ahora queremos anotar que el grado de digitalización de las obras de *Mahler* ha pasado a ser del 73%, tendremos que actualizar la relación tal y como se muestra en la figura 31.

Figura 31. Extensión de la relación después de una modificación

Compositions				
work	composer	yearComp	bCentury	digitDegree
Symphony 9	Mahler	1923	1897	~~70~~ 73
Symphony 5	Mahler	1918	1897	~~70~~ 73
Parsifal	Wagner	1857	1813	42

Observad que tendremos que modificar tantas tuplas como obras de *Mahler* tengamos en la relación.

El origen de las anomalías de inserción, eliminación y actualización que acabamos de ver es la mezcla de dos conceptos en una misma relación, lo cual implica además redundancia. La solución no es otra que la separación de conceptos, lo que se consigue dividiendo la relación original en dos nuevas relaciones.

La solución del caso utilizado como ejemplo vendría dada por la separación de los datos de obras y de compositores, lo que daría lugar al modelo normalizado siguiente:

```
Composer (cName, bCentury,
          digitDegree)

Work (wName, composer, yearComp)
     {composer} is foreign key to Composer
```

La extensión de estas nuevas relaciones es la que se presenta en la figura 32.

Figura 32. Extensión de dos relaciones normalizadas

Work		
wName	composer	yearComp
Symphony 9	Mahler	1923
Symphony 5	Mahler	1918
Parsifal	Wagner	1857
Conc Piano 3	Mozart	1779

Composer		
cName	bCentury	digitDegree
Mahler	19	70
Wagner	19	42
Mozart	18	42

Ahora podemos efectuar las operaciones de inserción, supresión y modificación que hemos utilizado en el ejemplo sin que se produzca ninguna anomalía.

La **teoría de la normalización** nos permitirá detectar si un diseño puede provocar anomalías como las que hemos descrito y, además, nos permitirá obtener un nuevo diseño con estas problemáticas resueltas.

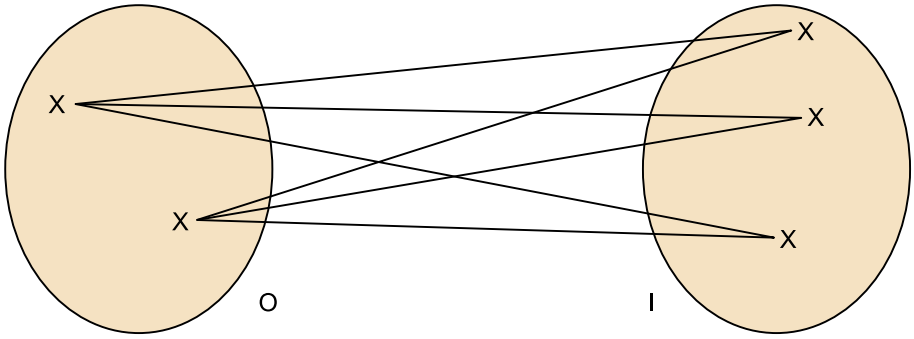
4.2. Conceptos previos

La teoría de la normalización se basa en el concepto de dependencia funcional, el cual a su vez se define en términos del álgebra relacional y de conceptos de la teoría de conjuntos. En este subapartado, se presentan estos elementos que permitirán abordar la teoría de la normalización.

Para empezar, repasaremos algunos conceptos de **álgebra de conjuntos**. Estos conceptos nos servirán después para razonar sobre los valores que toman los atributos de las relaciones. Concretamente, queremos definir el producto cartesiano, la correspondencia y la función:

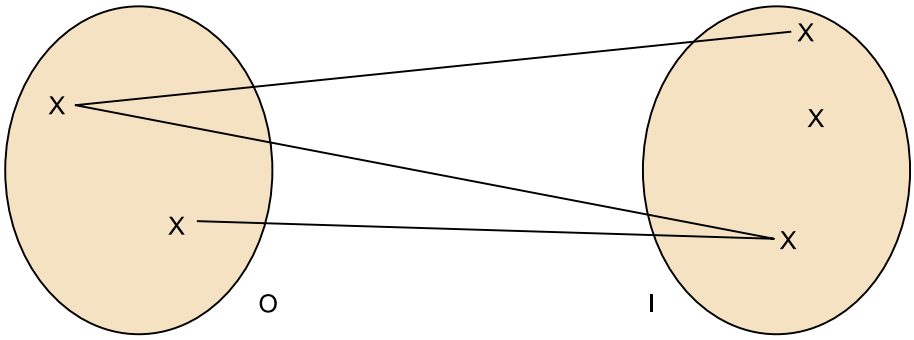
- Dados dos conjuntos O e I, el **producto cartesiano** $O \times I$ es el conjunto de todos los pares ordenados (o, i) tales que $o \in O$ e $i \in I$. Lo podemos representar gráficamente como se ve en la figura 33. El conjunto O recibe el nombre de conjunto *origen* y el conjunto I se denomina *conjunto imagen*.

Figura 33. El producto cartesiano



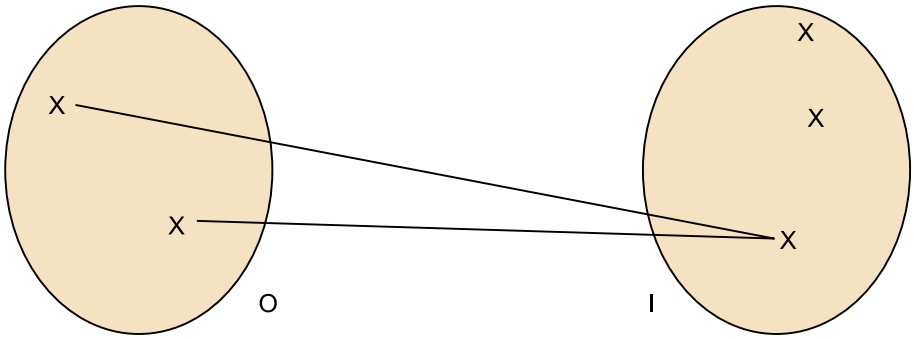
- Un subconjunto cualquiera del producto cartesiano es una **correspondencia**. La figura 34 muestra una correspondencia entre los mismos conjuntos del ejemplo anterior.

Figura 34. Una correspondencia



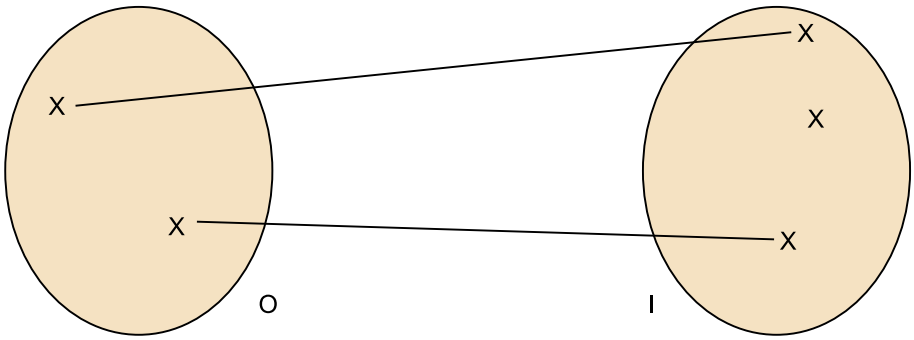
- Algunas correspondencias son, además, **funciones**: aquellas en las que cada elemento del conjunto origen está relacionado con un elemento (y solo uno) del conjunto imagen. La figura 35 es un ejemplo de función entre los conjuntos *O* e *I* de las figuras anteriores.

Figura 35. Una función



- Decimos que una función es **inyectiva** cuando cada elemento del conjunto imagen está relacionado, como mucho, con un elemento del conjunto origen. En la figura 36, podemos ver un ejemplo de función inyectiva.

Figura 36. Una función inyectiva



Como ya hemos visto, las relaciones se ajustan a un esquema que define sus atributos y tienen una extensión formada por tuplas que dan valores a los atributos.

Ved también

Podéis ver los conceptos previos del modelo relacional en el subapartado 3.1 de este módulo didáctico.

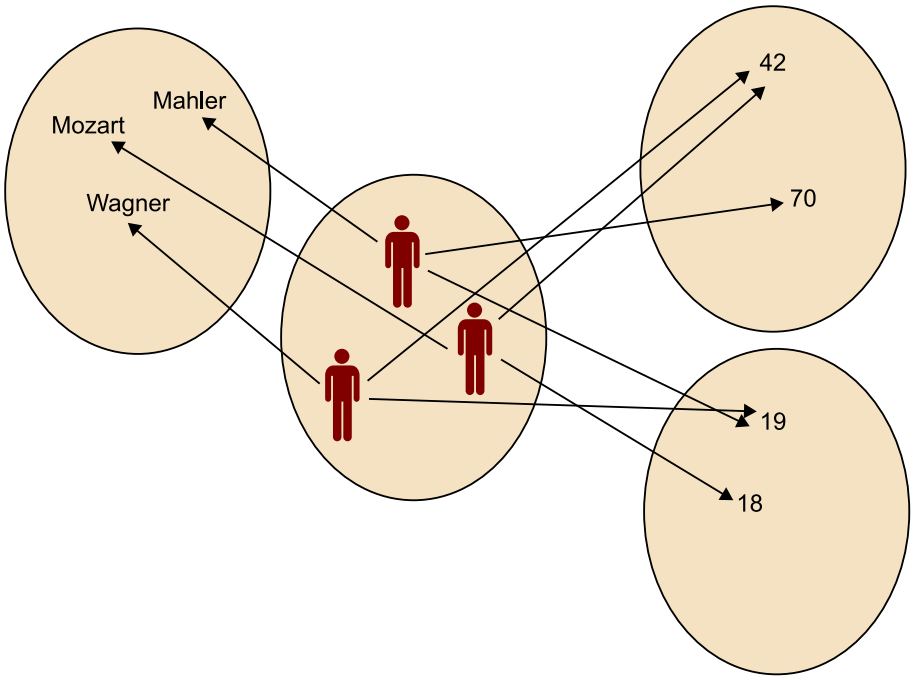
Cada tupla de la extensión de una relación tiene la información de una entidad del mundo real. Podemos usar el concepto de *función* para representar los valores que toma un atributo en cada tupla y, por lo tanto, para cada entidad.

Por ejemplo, la relación *Composer* de la figura 32 tiene tres atributos: *cName*, *Century* y *digitDegree*. El dominio del primero son las cadenas de caracteres y el de los otros dos, los números.

Los valores que toma un atributo en la extensión de una relación se pueden considerar una función que tiene como origen el conjunto de las entidades del mundo real y como imagen, el dominio del atributo.

La figura 37 es la visualización en términos de conjuntos y funciones de los atributos de la relación de la figura 32.

Figura 37. La visión como funciones de los atributos de una relación



Podemos observar que la función correspondiente a *cName* es inyectiva; fijémonos en que esta es una propiedad que cumplirán por definición todas las funciones que correspondan a atributos identificadores (observad que ya habíamos indicado que este atributo es una clave de la relación).

Las **dependencias funcionales** se establecen entre conjuntos de atributos de una relación, y las podemos considerar un tipo más de restricciones que deben satisfacer la extensión de la relación.

Dependencias funcionales

Dada una relación $R(A_1, A_2, ..., A_n)$, podemos declarar $X \rightarrow Y$ como dependencia funcional si $X, Y \subset \{A_1, A_2, ..., A_n\}$. Para satisfacer esta restricción, la extensión de la relación debe ser tal que X determine de manera única el valor de Y; es decir, que si dos tuplas tienen los mismos valores en los atributos de X, también tengan los mismos valores en los atributos de Y. En este caso, decimos que Y depende funcionalmente de X o, de modo alternativo, que X determina funcionalmente Y (y, por esto, X se denomina *determinante de la dependencia*). Es decir, si generalizamos el concepto de *función*, podemos decir que hay una función con origen X e imagen Y.

Observemos que un caso particular de dependencias funcionales son las claves de una relación: en este caso, los atributos que forman la clave determinan todos los demás. Esto es consecuencia de la no repetición de valores de la clave: puesto que sólo puede haber una tupla con un valor concreto de la clave, todas las tuplas con aquel valor (es decir, como máximo una) tienen el mismo valor para todos los demás atributos de la relación.

Ejemplo de dependencia funcional

Para acabar de presentar este concepto, lo ilustraremos con un ejemplo. Analicemos la relación siguiente para identificar dependencias funcionales:

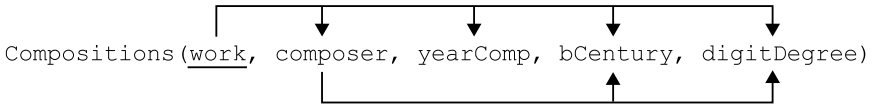
`Compositions(work, composer, yearComp, bCentury, digitDegree)`

Encontramos las dependencias funcionales siguientes:

- {*Work*} → {*composer*, *yearComp*, *bCentury*, *digitDegree*}. Esta dependencia corresponde a la clave primaria.
- {*composer*} → {*bCentury*, *digitDegree*}. Sabemos que el compositor determina el siglo en que nació y también el grado de digitalización conseguido en sus obras. Podemos comprobar que la extensión utilizada de ejemplo es correcta respecto a esta dependencia: Mahler, el compositor que se repite, aparece dos veces con los mismos valores para los atributos *bCentury* y *digitDegree*.

En cambio, {*composer*} → {*yearComp*} es falso, porque el compositor no determina el año de composición de las obras; un mismo compositor puede haber compuesto varias obras en años diferentes. Por este motivo, la extensión de la relación puede contener una tupla con Mahler y 1923 y otra con Mahler y 1918.

Para denotar las dependencias que hay en una relación, utilizaremos una notación basada en flechas, tal y como muestra el ejemplo siguiente:



Para finalizar, notemos que puede haber dependencias en las que podemos prescindir de algunos de los atributos del determinante.

Decimos que una dependencia funcional $X \rightarrow Y$ es completa cuando no hay ninguna otra dependencia $X' \rightarrow Y$, siendo X' un subconjunto propio de X.

Por ejemplo, podemos afirmar que {*Work*, *composer*} → {*yearComp*}, pero en esta dependencia podemos prescindir de *composer* porque {*Work*} → {*yearComp*} también es verdad. La primera de las dependencias anteriores no es completa, porque está la segunda y se verifica que {*Work*} es un subconjunto propio de {*Work*, *composer*}.

Reflexión

Observad que si en una dependencia funcional $X \rightarrow Y$ el conjunto X solo tiene un atributo, la dependencia funcional seguro que es completa.

4.3. Teoría de la normalización

El objetivo de la teoría de la normalización es fijar unas condiciones que nos garanticen la separación de conceptos y la ausencia de redundancia para evitar las anomalías de actualización. Estas condiciones se basan en gran medida en el concepto de dependencia funcional plena que acabamos de presentar en el subapartado anterior.

Las anomalías presentes en una relación tienen el origen en dependencias existentes entre los atributos de la relación. La **teoría de la normalización** define una serie de niveles denominados **formas normales** que eliminan progresivamente determinadas dependencias que son causantes de diferentes anomalías. Estas formas normales son inclusivas; es decir, si una relación cumple las condiciones de un determinado nivel, también cumple las condiciones de todos los niveles anteriores. Cuanto más alto es el grado de normalización, más redundancias se eliminan y, por lo tanto, menos anomalías se pueden producir.

La teoría de la normalización establece las bases para modificar una relación que no está en una determinada forma normal con el objetivo de que lo esté. Este proceso de modificación se denomina *normalización*. Tal como iremos viendo, una misma relación no normalizada se puede modificar de varias maneras hasta convertirse en una relación normalizada; es decir, el proceso de normalización puede tener diversos resultados.

Estudiaremos seis formas normales:

- la primera (1FN) se define en términos de la atomicidad de los atributos,

- las tres siguientes (2FN, 3FN y BCNF), en términos de dependencias funcionales,
- la penúltima (4FN) se basa en dependencias multivaluadas, y
- la última, (5FN), en la dependencia de proyección-combinación.

Estos dos últimos tipos de dependencias se presentarán en el momento de definir las formas normales correspondientes.

En un primer bloque trataremos las cuatro primeras formas normales, y después presentaremos un par de algoritmos capaces de normalizar hasta este nivel. Finalmente, veremos las dos últimas formas normales.

4.3.1. Primera forma normal

Una relación está en **primera forma normal** (**1FN**) si, y solo si, ningún atributo de la relación es en sí mismo una relación, ni descomponible ni con multiplicidad de valores. Los atributos, pues, deben ser atómicos.

Para ilustrar este concepto, fijémonos en la figura 38, en la que tenemos una relación que permite incluir información de los compositores.

Figura 38. Una relación con atributos no atómicos

Composers			
composer	works	yearComp	bCentury
Mahler	Symphony 9 Symphony 5	1923 1918	19
Wagner	Parsifal	1857	19

Esta relación no está en 1FN porque hay dos atributos, *Works* y *yearComp*, que no son atómicos. Para normalizar una relación en la primera forma normal, debemos aplanar los atributos que no son atómicos.

Aplanar un atributo no atómico

Aplanar un atributo no atómico de una relación consiste en sustituir cada tupla por tantas como repeticiones haya del atributo no atómico.

A partir de la relación de la figura 38, obtenemos la relación de la figura 39.

Figura 39. La relación después de aplanar los atributos

Composers			
composer	work	yearComp	bCentury
Mahler	Symphony 5	1918	19
Mahler	Symphony 9	1923	19
Wagner	Parsifal	1857	19

Debe observarse que la clave primaria de la relación normalizada cambia: tenemos que buscar la nueva clave a partir de la superclave formada por la composición de la que teníamos antes (en nuestro caso, *composer*) con la clave de la subrelación que formaban los atributos no atómicos (en nuestro caso, *Work* y *yearComp* que tiene como clave *Work*).

En nuestro ejemplo, la superclave es {*composer*, *Work*} pero dado que {*Work*} → {*composer*}, la clave primaria está formada exclusivamente por el atributo *Work*.

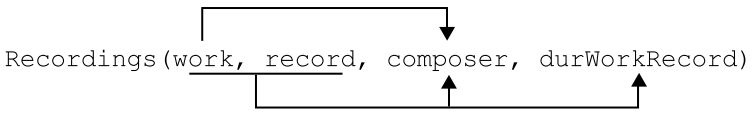
Hay que tener presente que el modelo relacional y los SGBD relacionales ya garantizan esta primera forma normal en cualquier relación que podamos definir.

4.3.2. Segunda forma normal

Una relación está en **segunda forma normal** (**2FN**) si, y solo si, está en primera forma normal y todo atributo que no forma parte de una clave candidata depende completamente de todas las claves candidatas de la relación.

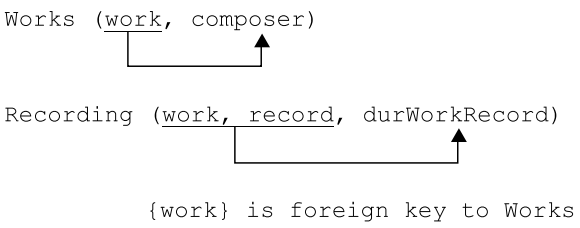
Observad que toda relación en primera forma normal que tenga las claves candidatas formadas por un solo atributo está de manera automática en segunda forma normal: por definición de clave, todo atributo depende de las claves candidatas y, en el caso de ser de un solo atributo, seguro que la dependencia es completa. Desde el momento en el que un determinante de una dependencia contiene solo una parte de alguna de las claves de la relación, esta ya no se encuentra en segunda forma normal.

Como ejemplo de esta problemática, consideremos la relación siguiente, que almacena las obras que contiene en cada una de las grabaciones (discos, cintas, archivos Mp3, etc.) que tenemos. Una misma obra (*Work*) puede estar repetida en varias grabaciones (*recordings*) y una grabación puede contener varias obras. También se desea guardar el compositor de cada obra (*composer*) y los minutos que dura cada obra (*durWorkRecord*) en una grabación determinada.



Esta relación, que tiene como clave primaria {*Work*, *record*}, no está en segunda forma normal porque *composer* no depende completamente de la clave, puesto que existe la dependencia {*Work*} → {*composer*}. El ejemplo nos permite ver con claridad cuál es el objetivo de la segunda forma normal: impedir que se mezclen dos hechos elementales que comparten parte de la clave (las grabaciones de las obras que tenemos, por un lado, y los compositores de las obras, por otro) en una misma relación. De este modo, evitaremos redundancias como

la de tener replicado el compositor de una obra tantas veces como el número de grabaciones que contienen la obra. Para normalizar una relación a 2FN debemos separar los hechos mezclados que violan la condición de 2FN en dos relaciones. En el ejemplo que estamos viendo, hay que transformar la relación *Recordings* en dos nuevas relaciones:



Como se puede observar, las dos relaciones resultantes se deben relacionar mediante una clave foránea por medio de la parte de la clave compartida por ambos hechos. Es preciso no confundir las flechas que representan dependencias con la que representa esta clave foránea.

Otros ejemplos de relaciones que no cumplirían la segunda forma normal serían los siguientes:

a)

La relación anterior, que tiene como clave primaria {*attr1, attr2*} y como clave alternativa {*attr3*}, no cumple la segunda forma normal: tenemos la dependencia {*attr2, attr3*} → {*attr4*} pero como podemos ver, aunque el determinante incluye la clave alternativa {*attr3*} al completo, solo contiene una parte de la calve primaria (*attr2*).

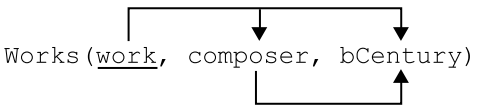
b)

Fijaos que en esta relación, aunque tenga los mismos atributos y/o columnas, no tiene clave alternativa {*attr3*}. Esta relación tampoco está en segunda forma normal porque el determinante del atributo {*attr4*} sigue incluyendo solo una parte de la clave primaria.

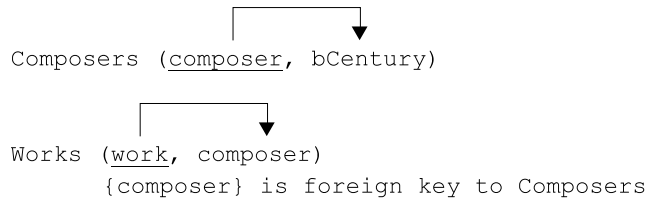
4.3.3. Tercera forma normal

Una relación está en **tercera forma normal** (**3FN**) si, y solo si, está en segunda forma normal y ningún atributo que no forma parte de una clave candidata depende de un conjunto de atributos que contiene alguno que no forma parte de una clave candidata.

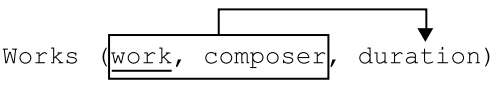
Podemos tomar la relación *Works* del ejemplo anterior y añadirle el siglo de nacimiento de los compositores (*bCentury*); obtenemos:



Esta relación, cuya clave es *Work*, está en 2FN, pero no está en 3FN porque, dada la dependencia {*composer*} → {*bCentury*}, ninguno de estos atributos forma parte de alguna clave candidata. Con la tercera forma normal evitamos, pues, que se mezclen hechos (quién es el compositor de una obra y cuándo nació un compositor, en el ejemplo) aunque compartan atributos que son clave en un hecho (*composer*, en el ejemplo) y otros que no lo son. De este modo, evitaremos redundancias, como la repetición del siglo de nacimiento de un compositor tantas veces como obras del compositor aparezcan en la relación. Para normalizar a 3FN, como antes, debemos separar el hecho que corresponde a la dependencia que viola la condición de 3FN. Aplicado al ejemplo anterior, obtendremos:



Ahora imaginemos que, en lugar de añadir el siglo de nacimiento de los compositores, añadimos la duración de la obra, que puede ser distinta en función del compositor:



En este caso nos encontramos con que, si bien el atributo *duration* está determinado por la clave {*work*} al completo (y, por lo tanto, la relación se encontraría en 2FN), el determinante también incluye el atributo {*composer*}, que no pertenece a ninguna clave. Esto implica que la relación no se encuentra en 3FN.

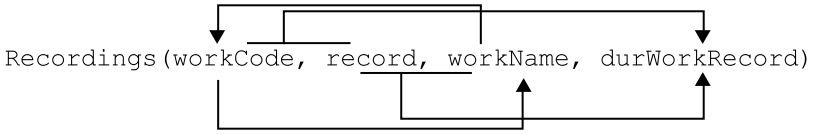
4.3.4. Forma normal de Boyce-Codd

Una relación está en la **forma normal de Boyce-Codd** (**FNBC**) si, y solo si, está en 3FN y los determinantes de todas las dependencias que presenta la relación son claves candidatas de esta.

La forma normal de Boyce-Codd

Esta forma normal se tuvo que definir (lo hicieron Boyce y Codd en 1974) para corregir algunas carencias de la 3FN que, cuando Codd la enunció en 1970, pensaba que bastaba para evitar las anomalías de actualización.

Para ver un ejemplo, podemos fijarnos en la siguiente relación, que está en 3FN:



Se trata de una relación similar a la del ejemplo anterior, a la que hemos añadido una nueva manera de identificar las obras. Ahora lo podemos hacer con un nombre y un código. La relación presenta redundancia porque repetimos tanto el nombre como el código de cada obra tantas veces como grabaciones tenga la obra. Cuando se definió la 3FN no se previó una situación como esta, en la que hay dos claves candidatas compuestas, solapadas y con dependencias entre partes de estas claves.

Tal y como hemos constatado gráficamente, existen cuatro dependencias en esta relación:

- {*workCode*} → {*workName*}
- {*workName*} → {*workCode*}
- {*workCode*, *record*} → {*durWorkRecord*}
- {*workName*, *record*} → {*durWorkRecord*}

Los determinantes de las dos últimas son claves candidatas de la relación, pero los determinantes de las dos primeras, no. Por lo tanto, la relación no está en FNBC. La figura 40 muestra una posible extensión de la relación.

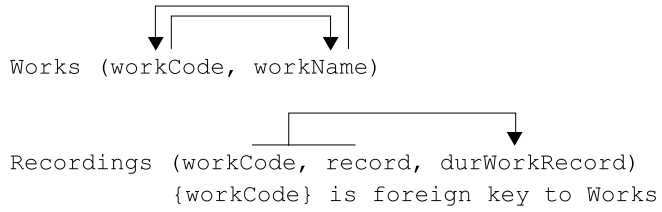
Figura 40. Ejemplo de relación que cumple la 3FN pero no la FNBC

Recordings			
workCode	record	workName	durWorkRecord
MahS5	DeutscheGrammophon001	Symphony 5	52
MahS9	Decca036	Symphony 9	43
MahS5	Naxos201	Symphony 5	55
MahS5	Sony187	Symphony 5	51

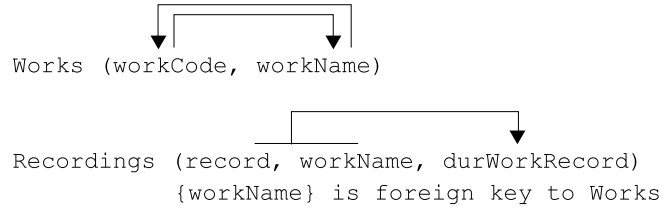
Observad que, efectivamente, se pueden producir anomalías. Por ejemplo, si queremos cambiar el nombre de la *Sinfonía 5*, tendremos que modificar tres tuplas.

Para normalizar en la FNBC tenemos que eliminar las dependencias cuyo determinante no constituye una clave candidata. Puesto que hay dos, podemos hacerlo de dos maneras:

1) Eliminando {*workCode*} → {*workName*}



2) Eliminando {*workName*} → {*workCode*}



Nos decidiremos por una opción u otra en función de si preferimos acceder a través del código o del nombre de la obra. Independientemente de la opción seleccionada, será preciso elegir cuál es la clave primaria de la relación *Works*, que tiene dos claves candidatas. Lo más coherente será elegir como clave primaria el atributo que es referenciado desde la relación *Recordings* (*workCode* si hemos elegido la opción 1 y *workName* si hemos elegido la opción 2).

4.3.5. Reglas de Armstrong

Tal y como hemos comentado, vistas las cuatro primeras formas normales, presentaremos dos algoritmos capaces de normalizar un esquema relacional hasta FNBC. Estos algoritmos se basan en una serie de propiedades que tienen las dependencias funcionales, que nos permiten deducir propiedades nuevas a partir de las que ya conocemos. Pongamos por ejemplo la relación que hemos utilizado para presentar las dependencias funcionales:

```
Compositions(work, composer, yearComp, bCentury, digitDegree)
```

Antes hemos mencionado que *Work* es su clave, pero también podríamos haberlo deducido así:

- {*Work*} → {*composer*, *yearComp*}, podemos afirmarlo por el conocimiento que tenemos del dominio sobre el que hacemos el diseño.

- Asimismo, como conocedores del dominio, sabemos que {composer} → {*bCentury*, *digitDegree*}.

- Y ahora podemos razonar que si *Work* determina *composer*, *Work* también determina los atributos determinados por *composer*, y deducimos que *Work* determina todos los demás atributos.

Las **reglas de deducción de Armstrong** que permiten hacer este y otros razonamientos son las que se enumeran a continuación:

Reflexión

Hay que aclarar que este no es un conjunto mínimo de reglas, puesto que algunas se pueden deducir a partir de las otras. En todo caso, tampoco hay un conjunto mínimo único, sino que podemos elegir varios conjuntos como axiomas y demostrar las otras reglas a partir de estos axiomas.

- Reflexividad: $X \rightarrow X$
- Aumentatividad: si $X \rightarrow Y$, entonces $X \cup Z \rightarrow Y$
- Distributividad: si $X \rightarrow Y \cup Z$, entonces $X \rightarrow Y$ y $X \rightarrow Z$
- Aditividad: si $X \rightarrow Y$ y $X \rightarrow Z$, entonces $X \rightarrow Y \cup Z$
- Transitividad: si $X \rightarrow Y$ y $Y \rightarrow Z$, entonces $X \rightarrow Z$
- Seudotransitividad: si $X \rightarrow Y$ y $Y \cup Z \rightarrow W$, entonces $X \cup Z \rightarrow W$

La **clausura transitiva** de un conjunto de dependencias D es el conjunto que se obtiene si se aplican repetidamente y de manera exhaustiva (hasta que ya no se pueden deducir nuevas dependencias) las reglas de Armstrong. La clausura transitiva de D, que denotamos con D^+, contiene todas las dependencias que son consecuencia de D y solo estas.

Por este motivo, la clausura de un conjunto de dependencias nos sirve para:

Reflexión

La normalización hasta FNBC es el grado de normalización mínimo que se requiere si no queremos que nuestra base de datos resulte afectada por anomalías.

- Confirmar o descartar una dependencia que sospechamos que se verifica.
- Encontrar todas las claves candidatas de las relaciones. Si queremos normalizar hasta FNBC, esta información es imprescindible.

- Confirmar o descartar que dos esquemas lógicos son equivalentes. A partir de las dependencias conocidas de D_1 y D_2, se puede decir que D_1 y D_2 son equivalentes si se verifica que $D_1^+ = D_2^+$.

Tomando como base algunas o todas estas reglas, se han definido algoritmos de normalización.

A continuación, presentamos dos algoritmos que se pueden ver como una aproximación a la generación automática de un esquema lógico a partir de los atributos y las dependencias de un dominio que surgen de una actividad previa equivalente al diseño conceptual de la BD. El primer algoritmo sigue un enfoque descendente, de descomposición de una relación potencialmente

muy grande que representa toda la información del dominio, mientras que el segundo está planteado desde un punto de vista ascendente, de agrupación de pequeñas informaciones elementales.

4.3.6. Algoritmo de análisis

La idea del algoritmo de análisis parte de una única relación, denominada *relación universal*, que contiene todos los atributos identificados. Utilizando las dependencias que también se deben haber identificado previamente, el algoritmo secciona repetidamente la relación universal hasta obtener un conjunto de relaciones normalizado. En cada paso de la partición, se comprueba si hay alguna relación que no está en la FNBC. Si no se encuentra ninguna, es que hemos conseguido un esquema normalizado; de lo contrario, se elige una de las relaciones que no están en la FNBC y se divide en dos usando la dependencia (o una cualquiera, si hay varias) que viola la FNBC en aquella relación. Este proceso se repite hasta que llegamos a la normalización. De hecho, el proceso es el que seguimos intuitivamente cuando normalizamos un esquema, con la diferencia de que habitualmente no partimos de una relación universal sino de un conjunto de relaciones resultantes de un diseño previo.

4.3.7. Algoritmo de síntesis

En este otro algoritmo, se sigue un enfoque ascendente. Partiendo de múltiples relaciones pequeñas, el algoritmo las fusiona para encontrar un conjunto de relaciones normalizado y tan compacto como sea posible. Estas relaciones iniciales provienen de lo que se denomina *recubrimiento mínimo de las dependencias funcionales*, que se define como un conjunto de dependencias más simple a partir del cual se pueden deducir las dependencias iniciales. El recubrimiento mínimo se genera en tres fases:

1) Desdoblar las dependencias de manera que sólo haya un atributo en la parte derecha.

2) Simplificar los determinantes eliminando atributos superfluos. Podremos eliminar los atributos que vienen determinados por otros atributos del mismo determinante en virtud de las dependencias.

3) Eliminar dependencias redundantes (dependencias que se pueden deducir a partir de otras dependencias).

Cada una de las dependencias del recubrimiento mínimo da lugar a una de las múltiples relaciones iniciales; y estas relaciones se van fusionando, agrupando las que comparten una clave candidata.

Ejemplo de obtención de un algoritmo de síntesis

Tomemos, por ejemplo, las dependencias siguientes:

- $\{codeWork\} \rightarrow \{workName\}$, $\{workName\} \rightarrow \{codeWork\}$

- {*codeWork*, *workName*} → {*Composer*, *centuryB*}
- {*Composer*} → {*centuryB*}
- {*codeWork*, *rec*} → {*durWorkRec*}

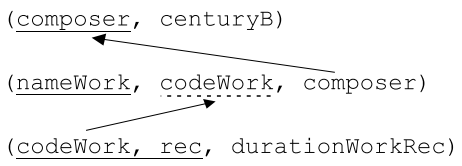
obtenemos un recubrimiento mínimo siguiendo los pasos que hemos descrito:

1) Tenemos que desdoblar {*codeWork*, *workName*} → {*Composer*, *centuryB*} en {*codeWork*, *workName*} → {*Composer*} y {*codeWork*, *workName*} → {*centuryB*}.

2) Los determinantes de las dos relaciones obtenidas se pueden simplificar. El resultado, entre otros posibles, es {*codeWork*} → {*Composer*} y {*codeWork*} → {*centuryB*}.

3) Ahora podemos detectar que tenemos una dependencia redundante: {*codeWork*} → {*centuryB*}, que se puede deducir a partir de {*codeWork*} → {*Composer*} y {*Composer*} → {*centuryB*}. El recubrimiento mínimo que obtenemos al finalizar este tercer paso está formado por estas dependencias: {*codeWork*} → {*workName*}, {*workName*} → {*codeWork*}, {*codeWork*} → {*Composer*}, {*Composer*} → {*centuryB*} y {*codeWork*, *rec*} → {*durWorkRec*}.

Las dependencias del recubrimiento han originado estas relaciones iniciales: (*codeWork*, *workName*), (*workName*, *codeWork*), (*codeWork*, *Composer*), (*Composer*, *centuryB*) y (*codeWork*, *rec*, *durWorkRec*). Las tres primeras relaciones comparten una clave candidata: *codeWork*. Se fusionarán y el esquema final obtenido por el algoritmo es:



4.3.8. Cuarta forma normal

La condición que define esta forma normal no se basa en dependencias funcionales sobre hechos monovaluados, sino que se basa en dependencias sobre hechos multivaluados que, si no se trasladan correctamente al esquema lógico, originan anomalías en relaciones que están en la FNBC.

Reflexión

Los hechos multivaluados aparecen con naturalidad en un esquema conceptual como tipo de relación con multiplicidad *, pero no se pueden representar como atributos multivaluados en el esquema lógico puesto el modelo relacional no admite atributos no atómicos.

Si intentamos transformar un atributo multivaluado del esquema conceptual en un atributo no atómico en un esquema relacional pensando que ya lo normalizaremos posteriormente, podemos obtener una relación en la FNBC que, sin embargo, presenta anomalías puesto que no está en 4FN.

Veamos todo esto con un ejemplo: queremos saber qué orquestas y qué obras ha dirigido cada director. Se trata de dos hechos multivaluados (un director puede haber dirigido muchas orquestas y puede haber dirigido muchas obras). Si nos imaginamos una relación que admite atributos no atómicos, podemos pensar en una situación como la que describe la figura 41.

Figura 41. Una relación con atributos no atómicos

Conductions		
Conductor	Works	Orchestras
C. Abbado	Symphony 9 Symphony 5	OCB London SO
H. von Karajan	Symphony 9 Conc Piano 1	Berliner Ph OCB

Puesto que sabemos que esto lo podemos imaginar pero no lo podemos definir en un SGBD relacional, aplanamos la relación para obtener la relación que describe la figura 42. Esta relación no tiene ninguna dependencia, aparte de la dependencia trivial reflexiva {*Conductor*, *Works*, *Orchestras*} → {*Conductor*, *Works*, *Orchestras*} y similares.

Figura 42. La relación después de aplanar los atributos

Conductions		
Conductor	Works	Orchestras
C. Abbado	Symphony 9	OCB
C. Abbado	Symphony 9	London SO
C. Abbado	Symphony 5	OCB
C. Abbado	Symphony 5	London SO
H. von Karajan	Symphony 9	Berliner Ph
H. von Karajan	Symphony 9	OCB
H. von Karajan	Conc Piano 1	Berliner Ph
H. von Karajan	Conc Piano 1	OCB

La relación está, pues, en la FNBC. Está claro, sin embargo, que esconde redundancias que provocan anomalías. Por ejemplo, si queremos registrar que Claudio Abbado ha dirigido la Filarmónica de Berlín, tendremos que añadir dos tuplas:

C. Abbado	Symphony 9	Berliner Ph
C. Abbado	Symphony 5	Berliner Ph

El problema, como en los casos presentados en las formas normales anteriores, es que en una única relación estamos mezclando dos hechos: las orquestas dirigidas y las obras dirigidas. La condición que revela esta mezcla de hechos es lo que denominamos *dependencia multivaluada independiente*, que definimos formalmente a continuación.

Sean X e Y subconjuntos de los atributos de una relación con el conjunto de atributos R. La dependencia **multivaluada independiente** de Y respecto de X, que denotamos por $X \rightarrow\rightarrow Y$, se verifica si para todo par de tuplas de la relación que tienen el mismo valor en X y diferente en Y, hay dos tuplas como estas que intercambian los valores de $R - X - Y$. Es decir, si existen $<x, y_1, z_1>$ y $<x, y_2, z_2>$, también deben existir $<x, y_1, z_2>$ y $<x, y_2, z_1>$ (siendo x valores posibles para X, y_1 e y_2 valores posibles para Y y z_1 y z_2 valores posibles para $R - X - Y$). Cuando se verifica $X \rightarrow\rightarrow Y$ también se verifica $X \rightarrow\rightarrow R - Y$.

Efectivamente, pues, tenemos las dependencias {*conductor*} →→ {*works*} y {*conductor*} →→ {*orchestras*} en la relación *Conductions*. Por ejemplo, para las tuplas <*C. Abbado*, *Sinfonía 9*, *London SO*> y <*C. Abbado*, *Sinfonía 5*, *OCB*> existen las tuplas <*C. Abbado*, *Sinfonía 9*, *OCB*> y <*C. Abbado*, *Sinfonía 5*, *London SO*>.

Reflexión

Observemos que las dependencias funcionales son un caso particular de las multivaluadas en el que un mismo valor de *X* solo puede aparecer con un solo valor de *Y*. Es decir, si {*X*} → {*Y*} entonces {*X*} →→ {*Y*}.

Ahora ya estamos en condiciones de definir la cuarta forma normal.

Una relación está en **cuarta forma normal** (**4FN**) si, y solo si, está en la FNBC y no presenta dependencias multivaluadas independientes.

Reflexión

Observad que por la definición de dependencia multivaluada independiente, una relación con solo uno o dos atributos no puede violar la condición de 4FN.

La relación *Conductions* no está en 4FN porque, como hemos visto, presenta dependencias multivaluadas independientes debidas a la mezcla de hechos en una misma relación. Por este motivo, la solución vuelve a consistir en la separación de los dos hechos en dos relaciones. La relación de la figura 42, una vez normalizada, se convierte en las que se presentan en la figura 43.

Figura 43. Dos relaciones normalizadas

ConductsWork	
Conductor	Work
C. Abbado	Symphony 9
C. Abbado	Symphony 5
H. von Karajan	Symphony 9
H. von Karajan	Conc Piano 1

ConductsOrc	
Conductor	Orchestra
C. Abbado	OCB
C. Abbado	London SO
H. von Karajan	OCB
H. von Karajan	Berliner Ph

Si recapitulamos el proceso que nos ha llevado hasta aquí, recordaremos que todo ha empezado con unos atributos multivaluados que hemos representado en una sola relación.

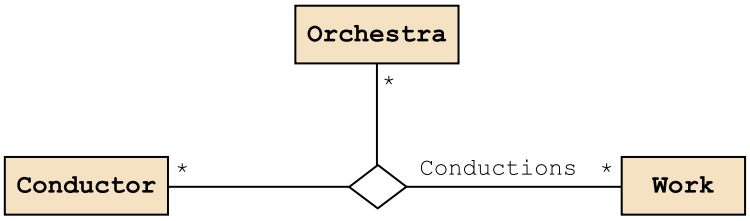
Si hubiéramos hecho una traducción sistemática del esquema conceptual, habríamos obtenido directamente el esquema lógico correcto.

El esquema conceptual de partida se muestra en la figura 44.

Figura 44. Los tipos de relaciones binarias de donde provienen las relaciones

La traducción de los tipos de relación *ConductsWork* y *ConductsOrc* lleva directamente a las relaciones normalizadas de la figura 43. La relación no normalizada de la figura 42 podría ser el resultado de traducir un esquema conceptual diferente, que presentamos en la figura 45.

Figura 45. El tipo de relación ternaria de donde proviene la relación



Sin embargo, entonces no existirían las dependencias multivaluadas porque ahora lo que queremos es registrar un único hecho ternario: qué obras ha dirigido cada director y con qué orquesta ha dirigido cada obra. En este caso, la relación *Conductions*, al no presentar las dependencias multivaluadas, estaría en 4FN. Y ahora podría tener una extensión como la que se muestra en la figura 46, que antes no era válida de acuerdo con las dependencias.

Figura 46. La extensión de la relación correspondiente al tipo de relación ternaria

Conductions		
Conductor	Work	Orchestra
C. Abbado	Symphony 9	OCB
C. Abbado	Symphony 5	London SO
H. von Karajan	Symphony 9	Berliner Ph
H. von Karajan	Conc Piano 1	OCB

Esta relación no presenta anomalías, porque ahora no tiene sentido decir que queremos añadir, por ejemplo, que Claudio Abbado ha dirigido la Filarmónica de Berlín. Ahora el hecho básico es que Claudio Abbado ha dirigido la Filarmónica de Berlín interpretando una obra concreta.

4.3.9. Quinta forma normal

Hay todavía un último caso en el que las tablas mezclan hechos y que la 4FN no detecta. Tomemos el caso de la figura 46 y supongamos que hay una ley de simetría en el dominio que afirma que si un director ha dirigido una obra y esta obra la ha interpretado una orquesta y el director ha dirigido esta orquesta, entonces el director ha dirigido la obra con esta orquesta. En este caso, la extensión de la figura 46 no cumpliría esta ley; faltaría la tupla

H. von Karajan	Symphony 9	OCB

Esto es así porque *H. von Karajan* ha dirigido la *Sinfonía 9* (según la tercera tupla), la *OCB* ha interpretado la *Sinfonía 9* (según la primera tupla) y *H. von Karajan* ha dirigido la *OCB* (según la cuarta tupla). Por otro lado, añadir esta nueva tupla no implica que la relación deje de estar en 4FN.

En casos así, la relación vuelve a presentar anomalías. Por ejemplo, si un director pasa a dirigir una nueva obra, por la ley de simetría tenemos que añadir tantas tuplas como orquestas que han interpretado la obra dirigida por el director. Si intentamos eliminar las anomalías descomponiendo la relación en dos, tenemos tres opciones de descomposición. Observemos que ninguna de las opciones es válida:

a) La descomposición siguiente no es válida porque no permite saber qué orquestas han dirigido los directores (podríamos pensar que Claudio Abbado ha dirigido la Filarmónica de Berlín).

CondWork	
Conductor	Work
C. Abbado	Symphony 9
C. Abbado	Symphony 5
H. von Karajan	Symphony 9
H. von Karajan	Conc Piano 1

WorkOrc	
Work	Orchestra
Symphony 9	OCB
Symphony 5	London SO
Symphony 9	Berliner Ph
Conc Piano 1	OCB

b) La descomposición siguiente no es válida porque no permite saber qué obras ha interpretado cada orquesta (podríamos pensar que la Sinfónica de Londres ha interpretado la sinfonía 9).

CondWork	
Conductor	Work
C. Abbado	Symphony 9
C. Abbado	Symphony 5
H. von Karajan	Symphony 9
H. von Karajan	Conc Piano 1

CondOrc	
Conductor	Orchestra
C. Abbado	OCB
C. Abbado	London SO
H. von Karajan	Berliner Ph
H. von Karajan	OCB

c) La descomposición siguiente no es válida porque no permite saber qué obras han dirigido los directores (podríamos pensar que Claudio Abbado ha dirigido el concierto para piano 1).

WorkOrc	
Work	Orchestra
Symphony 9	OCB
Symphony 5	London SO
Symphony 9	Berliner Ph
Conc Piano 1	OCB

CondOrc	
Conductor	Orchestra
C. Abbado	OCB
C. Abbado	London SO
H. von Karajan	Berliner Ph
H. von Karajan	OCB

Observemos, pues, que no podemos representar tres hechos con dos relaciones. Sin embargo, ¿qué sucede si tomamos las tres relaciones *CondWork*, *WorkOrc* y *CondOrc*?

Si descomponemos la relación original en estas otras tres relaciones, tenemos la misma información, porque podemos obtener la relación original haciendo la combinación natural de las otras tres. Se puede demostrar que esto es así para

cualquier extensión que cumpla la ley de simetría. Esta propiedad se conoce con el nombre de **dependencia de proyección-combinación** y es en lo que se basa la definición de quinta forma normal:

> Una relación está en **quinta forma normal** (**5FN**) si, y solo si, está en 4FN y no presenta dependencias de proyección-combinación.

Reflexión

A la vista de la definición, podemos afirmar que una relación con solo uno o dos atributos no puede violar la condición de 5FN.

De este modo, para saber si una relación que está en 4FN también está en 5FN tenemos que proyectar y combinar las proyecciones, y a continuación comprobar si obtenemos la misma relación original. En caso contrario, podemos asegurar que no hay dependencia de proyección-combinación y que la relación está en 5FN. De lo contrario, tenemos que estudiar si existe una ley de simetría que suponga que para toda extensión de la relación original, el proceso de proyección y combinación nos volverá a dar la relación original o si esto solo sucede en algunos casos.

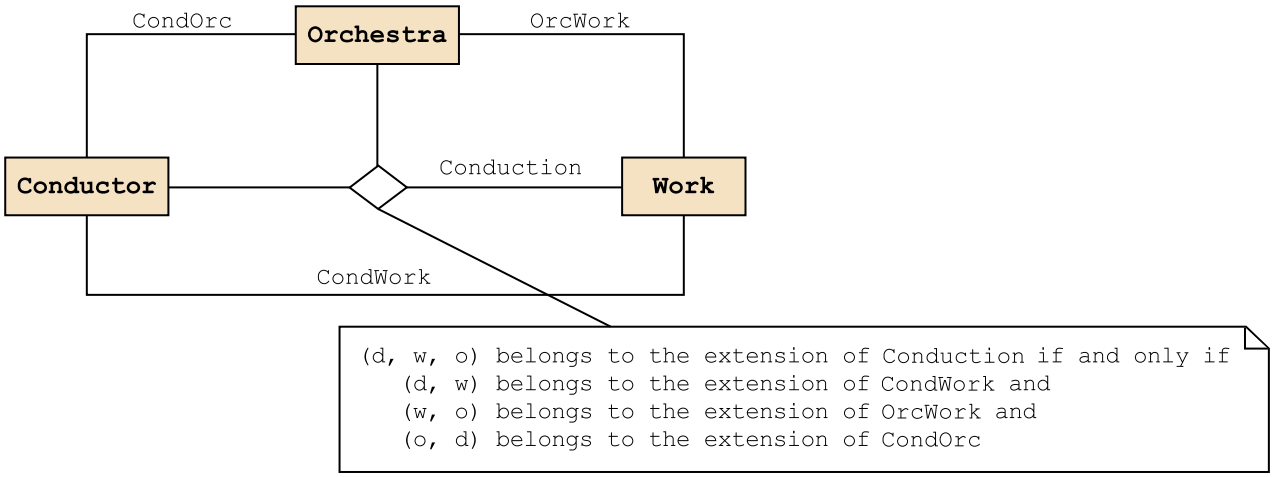
La relación *Conductions* en el momento de incorporar la ley de simetría no está en 5FN. Por esto, si partimos de la extensión de la figura 46 y le añadimos la tupla <*H. von Karajan*, *Sinfonia 9*, *OCB*> que ya hemos comentado que faltaba para cumplir esta ley, y luego proyectamos y combinamos, volvemos a la extensión de partida. En cambio, si ahora suponemos que no hay ley de simetría y que la extensión de *Conductions* es la de la figura 46, haciendo el proceso de proyección y combinación no obtendríamos la extensión de partida, sino la tupla adicional <*H. von Karajan*, *Sinfonía 9*, *OCB*>, y podríamos decir inmediatamente que la relación está en 5FN.

> Si retomamos el caso desde el comienzo, veremos que, si partimos del esquema conceptual correcto y lo traducimos de manera correcta al modelo relacional, obtendremos directamente el esquema lógico normalizado.

Cuando no se da la ley de dependencia, estamos ante un tipo de relación ternaria (la que se muestra en la figura 45), que por tanto, se convierte en una única relación normalizada. Cuando añadimos la ley de simetría, estamos introduciendo tres tipos de relaciones binarias y una ternaria que se deriva de las tres binarias, como se muestra en la figura 47.

A partir de este esquema conceptual, debemos traducir, no el tipo de relación derivado (que daría lugar a la relación que no está en 5FN), sino las tres binarias (que producen las tres relaciones que sí están en 5FN).

Figura 47. Los tipos de relaciones binarias de donde provienen las relaciones normalizadas y el tipo de relación ternaria derivada



4.4. Práctica de la normalización

El proceso de normalización tiene en general efectos beneficiosos para la base de datos y preserva la semántica del esquema de partida. El proceso siempre es factible y existen algoritmos que normalizan hasta la FNBC. Un mismo esquema de partida se puede normalizar en general de más de una manera (tanto si lo hacemos manualmente como si usamos algún algoritmo), y es trabajo del diseñador elegir la alternativa que más interesa en cada caso.

Los efectos beneficiosos de aplicar las formas normales son la eliminación de redundancias y anomalías y la clarificación del esquema, al separar los hechos diferentes que quizá se encuentran mezclados en el esquema inicial.

El momento y la oportunidad de la aplicación de la normalización pueden ser varios en función del contexto en el que se lleva a cabo el diseño de la base de datos. Si nos encontramos en una situación de desarrollo a partir de un sistema preexistente (o de una parte), puede ser interesante aplicar la normalización antes de empezar, si no la tenemos garantizada. Si nos encontramos modificando un esquema lógico cuya documentación correspondiente no tenemos en el esquema conceptual, la normalización *a posteriori* es la única manera de tener la certeza de que el resultado está normalizado. En cambio, si partimos de cero y disponemos de un esquema conceptual correcto, una traducción correcta al modelo relacional nos asegura un esquema lógico normalizado. Aun así, puesto que el esquema conceptual de partida puede ser incorrecto y en el proceso de traducción podemos cometer errores, es muy recomendable comprobar si el resultado final está normalizado. Por ejemplo, si hemos introducido en el esquema un tipo de relación ternaria en vez de dos o tres binarias, es posible que esto haya provocado la aparición de alguna relación que no está en 4FN o 5FN.

La normalización tiene como consecuencia no deseada la penalización de la recuperación de la base de datos. Puesto que descompone las relaciones, separa atributos que en un esquema no normalizado estarían en la misma relación. Esto implica la necesidad de ejecutar más operaciones de combinación para obtener información que en el esquema no normalizado lograríamos consultando una única relación.

En general, sin embargo, podemos asumir esta desventaja a cambio de eliminar las anomalías y facilitar la integridad de la base de datos. Hay, sin embargo, casos o partes de las bases de datos en los que las actualizaciones son poco frecuentes. En estos casos, el equilibrio entre ventajas e inconvenientes puede llevarnos a decantarnos por la opción de no normalizar.

La **desnormalización** es el proceso de deshacer la normalización agrupando datos lógicamente independientes o añadiendo redundancia a la base de datos con el objetivo de hacer más eficientes las consultas.

Hay que tener presente que este proceso penaliza las actualizaciones de la base de datos y requiere un diseño muy cuidadoso de las restricciones para garantizar que la base de datos se mantiene consistente. Se requiere, por lo tanto, un análisis cuidadoso de ventajas e inconvenientes de la desnormalización en cada caso. Un caso en el que la desnormalización sale más a cuenta es el de las bases de datos dedicadas solo a consultar datos históricos agregados por diferentes conceptos.

Una opción a medio camino de la desnormalización es la definición de vistas materializadas adicionales al diseño normalizado que ayuden a acelerar las consultas más frecuentes o costosas.

Resumen

En este módulo hemos estudiado el proceso de diseño lógico como una de las etapas del desarrollo de una base de datos para un sistema de información. Este proceso parte del esquema conceptual que hemos obtenido en la fase de especificación y genera como resultado el esquema lógico de la base de datos.

Hemos empezado el estudio identificando las trampas de diseño, posibles errores que pueden haberse cometido durante el diseño conceptual y que conviene repasar antes de tomarlo como punto de partida del diseño lógico.

Hemos tratado la etapa de diseño lógico en el caso particular de transformación de modelos conceptuales basados en el lenguaje UML en modelos lógicos relacionales. Hemos analizado cada uno de los elementos del esquema conceptual (tipos de entidades, tipos de relaciones, generalizaciones y tipos de entidades asociativas) y hemos ofrecido pautas de transformación para cada uno. Hemos separado cada elemento en los distintos casos en los que se puede dividir y hemos dado la solución para cada uno, sin dejar de lado las restricciones. En algunos casos, hemos presentado varias alternativas de transformación y hemos puesto énfasis en el hecho de que muchas veces la mejor alternativa será la que evite la presencia de valores nulos.

Finalmente, hemos estudiado la teoría de la normalización, que permite asegurar que el esquema relacional satisface una serie de condiciones que garantizan una mejor calidad de la base de datos. Hemos identificado las anomalías que se pueden producir en una base de datos no normalizada y hemos definido una serie de formas normales mediante condiciones que, si son satisfechas por la base de datos, nos garantizan la ausencia de anomalías. Hemos terminado con una reflexión sobre las ventajas y los inconvenientes de la normalización y una breve introducción al concepto de *desnormalización*.

Glosario

abrazo mortal de carga *f* Imposibilidad de insertar tuplas en ninguna tabla de un conjunto de tablas porque las claves foráneas que contienen forman un ciclo.

abrazo mortal de definición *f* Imposibilidad de definir un conjunto de tablas porque las claves foráneas que contienen forman un ciclo.

anomalía de actualización *f* Necesidad de actualizar muchas tuplas para reflejar un cambio elemental.

clausura transitiva de un conjunto de dependencias *f* Conjunto de todas las dependencias que se pueden deducir aplicando de manera reiterada las reglas de Armstrong a partir del conjunto inicial.

dependencia funcional *f* En una relación, decimos que un conjunto de atributos Y depende de un conjunto de atributos X ($X \rightarrow Y$) si siempre que dos tuplas tienen los mismos valores en los atributos de X, también tienen los mismos valores en los atributos de Y.

dependencia funcional completa *f* $X \rightarrow Y$ es completa si no existe ningún X' subconjunto propio de X tal que $X' \rightarrow Y$.

dependencia multivaluada independiente *f* Decimos que hay una dependencia multivaluada independiente de Y respecto a X, que denotamos por $X \twoheadrightarrow Y$, si se verifica que para todo par de tuplas de la relación que tienen el mismo valor en X y diferente en Y, hay dos tuplas como estas que intercambian los valores de $R - X - Y$.

dependencia proyección-combinación *f* Decimos que una relación con tres atributos presenta una dependencia de proyección-combinación si para cualquier extensión correcta de la relación se verifica que, como resultado de descomponerla en otras tres relaciones de dos atributos (haciendo las correspondientes proyecciones) y combinar estas tres relaciones a continuación, obtenemos nuevamente la relación original.

desnormalización *f* Proceso consistente en deshacer la normalización agrupando datos lógicamente independientes o añadiendo redundancia a la base de datos, con el objetivo de hacer más eficientes las consultas.

determinante *m* Conjunto de atributos X de una dependencia funcional $X \rightarrow Y$.

diseño lógico *m* Proceso de transformación de un esquema conceptual en un esquema lógico de base de datos.

forma normal *f* Decimos que una relación está en una determinada forma normal si satisface las condiciones fijadas por aquella forma normal. Las formas normales son inclusivas: la condición de una forma normal de nivel superior implica la condición de cada uno de los niveles inferiores y, por lo tanto, si una relación está en una forma normal también está en todas las formas normales de nivel inferior.

normalización *f* Proceso por el cual, a partir de un conjunto de relaciones, se obtiene un conjunto de relaciones equivalente que satisface la condición de la forma normal deseada.

trampa de diseño *m* Patrón del esquema conceptual que puede inducir a cometer errores en la interpretación del mundo real.

recubrimiento mínimo *m* Conjunto más simple de otro conjunto a partir del cual se pueden deducir las dependencias del conjunto original.

reglas de Armstrong *f pl* Conjunto de reglas de deducción que permiten demostrar si una dependencia es consecuencia de otras.

restricción *f* Condición que limita las extensiones válidas de una relación.

Bibliografía

Camps, R.; Cassany M. J.; Conesa, J.; Costal, D.; Figuls, D.; Martín, C.; Rius, A.; Rodríguez, M. E.; Urpí, T. (2011). *Uso de bases de datos*. FUOC.

Elmasri, Ramez; Navathe, Shamkant, B. (2007). *Fundamentos de sistemas de bases de datos* (5.ª ed.). Madrid: Pearson Educación.

Gulutzan, P.; Pelzer, T. (1999). *SQL-99 Complete, really*. R&D Books.

Ramakrishnan, Raghu; Gehrke, Johannes (2003). *Database management systems* (3.ª ed.). Boston: McGraw-Hill Higher Education.

Sciore, E. (2008). *Database Design and Implementation*. Wiley.

Unified Modeling Language: Superstructure (versión 2.0, OMG doc. formal/05-07-04). Accesible en línea desde la web www.omg.org.