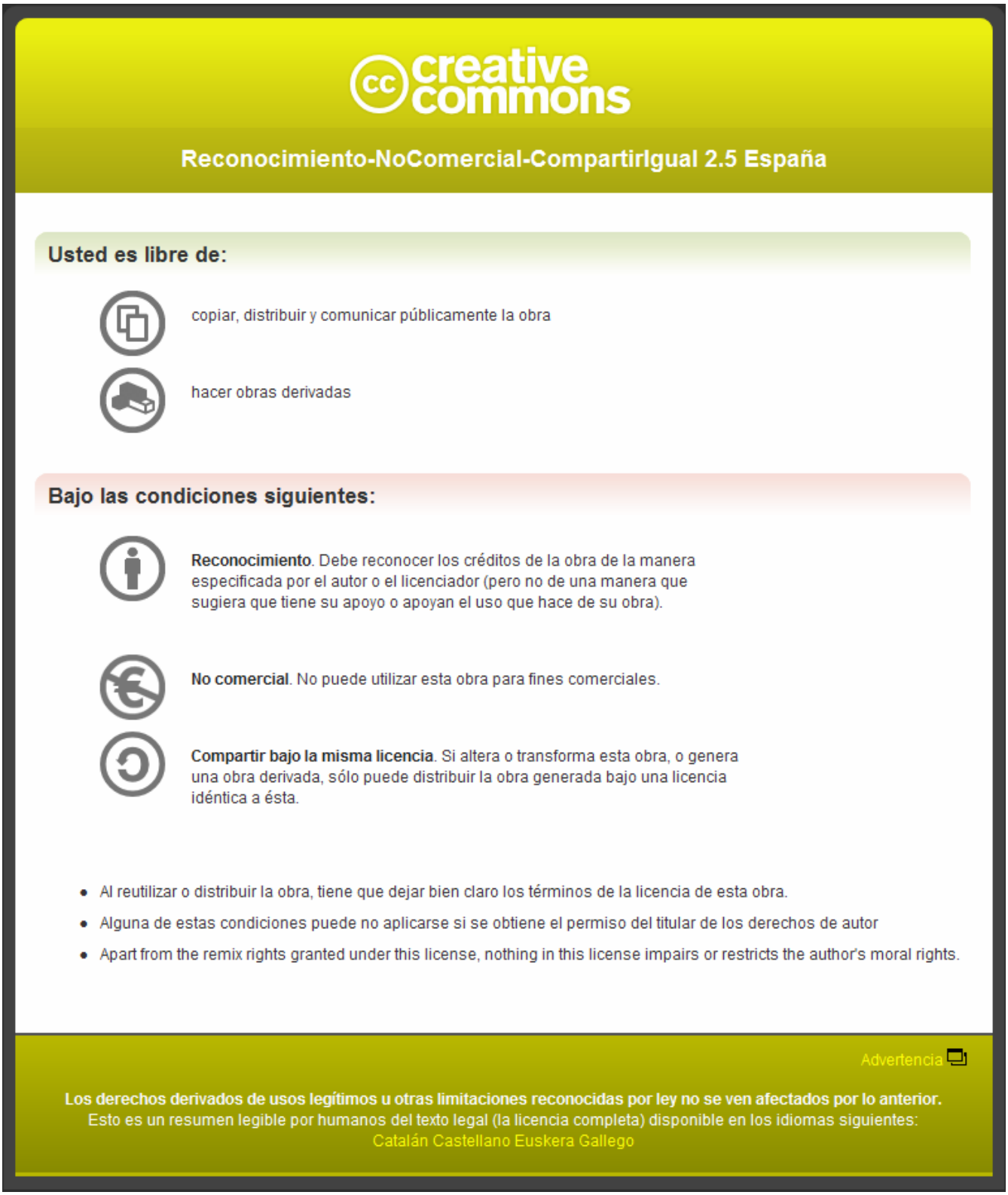
Unidad 2:
Bases de Datos Relacionales

Sistemas Gestores de Bases de Datos
2º de ASI

Autor: Jorge Sánchez Asenjo (año 2008) http://www.jorgesanchez.net
e-mail:info@jorgesanchez.net



Para ver una copia completa de la licencia, acudir a la dirección http://creativecommons.org/licenses/by-nc-sa/2.5/es/legalcode.es

(2) Bases de Datos Relacionales

esquema de la unidad

(2.1) el modelo relacional

(2.1.1) introducción

Edgar Frank Codd definió las bases del modelo relacional a finales de los 60. En 1970 publica el documento *"A Relational Model of data for Large Shared Data Banks"* ("*Un modelo relacional de datos para grandes bancos de datos compartidos*"). Actualmente se considera que ese es uno de los documentos más influyentes de toda la historia de la informática. Lo es porque en él se definieron las bases del llamado **Modelo Relacional de Bases de Datos.** Anteriormente el único modelo teórico estandarizado era el **Codasyl** que se utilizó masivamente en los años 70 como paradigma del modelo en red de bases de datos.

Codd se apoya en los trabajos de los matemáticos **Cantor** y **Childs** (cuya teoría de conjuntos es la verdadera base del modelo relacional). Según Codd los datos se agrupan en **relaciones** (actualmente llamadas **tablas**) que es un concepto que se refiere a la estructura que aglutina datos referidos a una misma entidad de forma independiente respecto a su almacenamiento físico.

Lo que Codd intentaba fundamentalmente es evitar que las usuarias y usuarios de la base de datos tuvieran que verse obligadas a aprender los entresijos internos del sistema. Pretendía que los usuarios/as trabajaran de forma sencilla e independiente del funcionamiento físico de la base de datos en sí. Fue un enfoque revolucionario.

Aunque trabajaba para **IBM**, esta empresa no recibió de buen grado sus teorías (de hecho continuó trabajando en su modelo en red **IMS**). De hecho fueron otras empresas (en especial **Oracle**) las que implementaron sus teorías. Pocos años después el modelo se empezó a utilizar cada vez más, hasta finalmente ser el modelo de bases de datos más popular. Hoy en día casi todas las bases de datos siguen este modelo.

(2.1.2) objetivos

Codd perseguía estos objetivos con su modelo:

- **Independencia física**. La forma de almacenar los datos, no debe influir en su manipulación lógica. Si la forma de almacenar los datos cambia, los usuarios no tienen siquiera porque percibirlo y seguirán trabajando de la misma forma con la base de datos. Esto permite que los usuarios y usuarias se concentren en qué quieren consultar en la base de datos y no en cómo está realizada la misma.
- **Independencia lógica**. Las aplicaciones que utilizan la base de datos no deben ser modificadas porque se modifiquen elementos de la base de datos. Es decir, añadir, borrar y suprimir datos, no influye en las vistas de los usuarios. De una manera más precisa, gracias a esta independencia el esquema externo de la base de datos es realmente independiente del modelo lógico.

- **Flexibilidad**. La base de datos ofrece fácilmente distintas vistas en función de los usuarios y aplicaciones.
- **Uniformidad**. Las estructuras lógicas siempre tienen una única forma conceptual (las tablas).
- **Sencillez**. Facilidad de manejo (algo cuestionable, pero ciertamente verdadero si comparamos con los sistemas gestores de bases de datos anteriores a este modelo).

(2.1.3) historia del modelo relacional

Año	Hecho
1970	Codd publica las bases del modelo relacional
1971-72	Primeros desarrollos teóricos
1973-78	Primeros prototipos de base de datos relacional. Son el **System R** de IBM. En ese sistema se desarrolla **Sequel** que con el tiempo cambiará su nombre a SQL.
1974	La Universidad de Berkeley desarrolla **Ingres**, SGBD relacional basado en cálculo relacional. Utilizaba el lenguaje **Quel** desarrollado en las universidades y muy popular en la época en ámbitos académicos.
1978	Aparece el lenguaje **QBE** (***Query By Example***) lenguaje de acceso relacional a los archivos **VSAM** de IBM
1979	Aparece **Oracle**, el primer SGBD comercial relacional (ganando en unas semanas al **System/38** de IBM). Implementa SQL y se convertirá en el sistema gestor de bases de datos relacionales líder del mercado. Codd revisa su modelo relacional y lanza el modelo **RM/T** como un intento de subsanar sus deficiencias.
1981	Aparece **Informix** como SGBD relacional para Unix
1983	Aparece **DB2**, el sistema gestor de bases de datos relacionales de IBM
1984	Aparece la base de datos **Sybase** que llegó a ser la segunda más popular (tras Oracle)
1986	ANSI normaliza el SQL (**SQL/ANSI**). SQL es ya de hecho el lenguaje principal de gestión de bases de datos relacionales.
1987	ISO también normaliza SQL. Es el **SQL ISO**(9075)
1988	La versión 6 de Oracle incorpora el lenguaje procedimental **PL/SQL**

Año	Hecho
1989	ISO revisa el estándar y publica el estándar **SQL Addendum.** **Microsoft** y **Sybase** desarrollan **SQL Server** para el sistema operativo **OS/2** de Microsoft e **IBM**. Durante años Sybase y SQL Server fueron el mismo producto.
1990	Versión dos del modelo relacional (**RM/V2**) realizada por Codd. Propuesta de **Michael Stonebraker** para añadir al modelo relacional capacidades de orientación a objetos.
1992	ISO publica el estándar **SQL 92** (todavía el más utilizado)
1995	Manifiesto de **Darwen** y **Date** en el que animan a reinterpretar el modelo relacional desde una perspectiva de objetos. Aparece el modelo objeto/relacional. Aparece **MySQL** una base de datos relacional de código abierto con licencia GNU que se hace muy popular entre los desarrolladores de páginas web.
1996	ANSI normaliza el lenguaje procedimental basado en SQL y lo llaman **SQL/PSM.** Permite técnicas propias de los lenguajes de programación estructurada. Aparece el SGBD abierto **PostgreSQL** como remodelación de la antigua Ingres, utilizando de forma nativa el lenguaje SQL (en lugar de Quel).
1999	ISO publica un nuevo estándar que incluye características más avanzadas. Se llama **SQL 99** (también se le conoce como SQL 200)
2003	ISO publica el estándar **SQL 2003.** En él se añade SQL/PSM al estándar.
2006	Estándar ISO. **SQL 2006**
2008	Estándar ISO. **SQL 2008**

(2.2) estructura de las bases de datos relacionales

(2.2.1) relación o tabla

Según el modelo relacional (desde que Codd lo enunció) el elemento fundamental es lo que se conoce como **relación**, aunque más habitualmente se le llama **tabla** (o también array o matriz). Codd definió las relaciones utilizando un lenguaje matemático, pero se pueden asociar a la idea de tabla (de filas y columnas) ya que es más fácil de entender.

No hay que confundir la idea de relación según el modelo de **Codd**, con lo que significa una relación en el modelo Entidad/Relación de **Chen.** No tienen nada que ver

Las relaciones constan de:

- **Atributos**. Referido a cada propiedad de los datos que se almacenan en la relación (nombre, dni,...).
- **Tuplas.** Referido a cada elemento de la relación. Por ejemplo si una relación almacena personas, una tupla representaría a una persona en concreto.

Puesto que una relación se representa como una tabla; podemos entender que las columnas de la tabla son los atributos; y las filas, las tuplas.

atributo 1	atributo 2	atributo 3	....	atributo n	
valor 1,1	valor 1,2	valor 1,3	....	valor 1,n	← tupla 1
valor 2,1	valor 2,2	valor 2,3	....	valor 2,n	← tupla 2
.....	.....	.....	....	.....	....
valor m,1	valor m,2	valor m,3	....	valor m,n	← tupla m

La tabla superior representa la estructura de una relación según el modelo de Codd.

(2.2.2) tupla

Cada una de las filas de la relación. Se corresponde con la idea clásica de **registro**. Representa por tanto cada elemento individual de esa relación. Tiene que cumplir que:

- Cada tupla se debe corresponder con un elemento del mundo real.
- No puede haber dos tuplas iguales (con todos los valores iguales).

(2.2.3) dominio

Un dominio contiene todos los posibles valores que puede tomar un determinado atributo. Dos atributos distintos pueden tener el mismo dominio.

Un dominio en realidad es un conjunto finito de valores del mismo tipo. A los dominios se les asigna un nombre y así podemos referirnos a ese nombre en más de un atributo.

La forma de indicar el contenido de un dominio se puede hacer utilizando dos posibles técnicas:

- **Intensión.** Se define el nomino indicando la definición exacta de sus posibles valores. Por intensión se puede definir el dominio de edades de los trabajadores como: *números enteros entre el 16 y el 65* (un trabajador sólo podría tener una edad entre 16 y 65 años).
- **Extensión**. Se indican algunos valores y se sobreentiende el resto gracias a que se autodefinen con los anteriores. Por ejemplo el dominio localidad se podría definir por extensión así: *Palencia, Valladolid, Villamuriel de Cerrato,...*

Además pueden ser:

- **Generales**. Los valores están comprendidos entre un máximo y un mínimo
- **Restringidos**. Sólo pueden tomar un conjunto de valores

(2.2.4) grado

Indica el tamaño de una relación en base al número de columnas (atributos) de la misma. Lógicamente cuanto mayor es el grado de una relación, mayor es su complejidad al manejarla.

(2.2.5) cardinalidad

Número de tuplas de una relación, o número de filas de una tabla.

(2.2.6) sinónimos

Los términos vistos anteriormente tienen distintos sinónimos según la nomenclatura utilizada. A ese respecto se utilizan tres nomenclaturas:

Términos 1 (nomenclatura relacional)		Términos 2 (nomenclatura tabla)		Términos 3 (nomenclatura ficheros)
relación	=	tabla	=	fichero
tupla	=	fila	=	registro
atributo	=	columna	=	campo
grado	=	nº de columnas	=	nº de campos
cardinalidad	=	nº de filas	=	nº de registros

(2.2.7) definición formal de relación

Una relación está formada por estos elementos:

- **Nombre.** Identifica la relación.
- **Cabecera de relación.** Conjunto de todos los pares atributo-domino de la relación:
  $\{(A_i:D_i)\}_{i=1}^{n}$ donde ***n*** es el **grado**.
- **Cuerpo de la relación.** Representa el conjunto de ***m*** tuplas $\{t_1, t_2,... t_n\}$ que forman la relación. Cada tupla es un conjunto de ***n*** pares atributo-valor $\{(A_i:V_{ij})\}$, donde V_{ij} es el valor $_j$ del dominio D_i asociado al atributo A_i.
- **Esquema de la relación.** Se forma con el nombre R y la cabecera. Es decir:
  $R\{(A_i:D_i)\}_{i=1}^{n}$
- **Estado de la relación.** Lo forman el esquema y el cuerpo.

Ejemplo:

Clientes		
DNI	**Nombre**	**Edad**
12333944C	Ana	52
12374678G	Eva	27
28238232H	Martín	33

Esquema: **Cliente**(DNI:**DNI,** Nombre:**Nombre**, Edad:**Edad)**

Cuerpo: **{**(DNI: "12333944C", Nombre:"Ana", Edad:52), (DNI: "12374678G", Nombre:"Eva", Edad;52), (DNI: "28238232H", Nombre:"Martín",Edad:33)}

(2.2.8) propiedades de las tablas (o relaciones)

- Cada tabla tiene un nombre distinto
- Cada atributo de la tabla toma un solo valor en cada tupla
- Cada atributo tiene un nombre distinto en cada tabla (aunque puede coincidir en tablas distintas)
- Cada tupla es única (no hay tuplas duplicadas)
- El orden de los atributos no importa
- El orden de las tuplas no importa

(2.2.9) tipos de tablas

- **Persistentes.** Sólo pueden ser borradas por los usuarios:
  - **Bases**. Independientes, se crean indicando su estructura y sus ejemplares. Contienen tanto datos como metadatos.
  - **Vistas**. Son tablas que sólo almacenan una definición de consulta, resultado de la cual se produce una tabla cuyos datos proceden de las bases o de otras vistas e instantáneas. Si los datos de las tablas base cambian, los de la vista que utiliza esos datos también cambia.
  - **Instantáneas**. Son vistas (creadas de la misma forma) que sí que almacenan los datos que muestra, además de la consulta que dio lugar a esa vista. Sólo modifican su resultado (actualizan los datos) siendo refrescadas por el sistema cada cierto tiempo (con lo que tienen el riesgo de que muestren algunos datos obsoletos.
- **Temporales**. Son tablas que se eliminan automáticamente por el sistema. Pueden ser de cualquiera de los tipos anteriores. Las utiliza el SGBD como almacén intermedio de datos (resultados de consultas, por ejemplo)

(2.2.10) claves

clave candidata

Conjunto de atributos que identifican unívocamente cada tupla de la relación. Es decir columnas cuyos valores no se repiten en ninguna otra tupla de esa tabla. Toda tabla en el modelo relacional debe tener al menos una clave candidata (puede incluso haber más)

clave primaria

Clave candidata que se escoge como **identificador** de las tuplas. Se elige como primaria la candidata que identifique mejor a cada tupla en el contexto de la base de datos.

Por ejemplo un campo con el *DNI* sería clave candidata de una tabla de clientes, si esa tabla tiene un campo de *código de cliente*, éste sería mejor candidato (y por lo tanto clave principal) porque es mejor identificador para ese contexto.

clave alternativa

Cualquier clave candidata que no sea primaria.

clave externa, ajena o secundaria

Son los datos de atributos de una tabla cuyos valores están relacionados con atributos de otra tabla. Por ejemplo en la tabla equipos tenemos estos datos:

Equipo	Nº Equipo
Real Madrid	1
F.C. Barcelona	2
Athletic Bilbao	3

En la tabla anterior la clave principal es el atributo **nº equipo.** En otra tabla tenemos:

Nº Jugador	Jugador	Nº Equipo
1	Karanka	3
2	Ronaldinho	2
3	Raul	1
4	Beckham	1

El atributo *Nº Equipo* sirve para relacionar el Jugador con el equipo al que pertenece. Ese campo en la tabla de jugadores es una clave secundaria.

nulos

En los lenguajes de programación se utiliza el valor nulo para reflejar que un identificador (una variable, un objeto,..) no tiene ningún contenido. Por ejemplo cuando un puntero en lenguaje C señala a **null** se dice que no está señalando a nadie. Al programar en esos lenguajes se trata de un valor que no permite utilizarse en operaciones aritméticas o lógicas.

Las bases de datos relacionales permiten más posibilidades para el valor nulo (**null**), aunque su significado no cambia: valor vacío. No obstante en las bases de datos se utiliza para diversos fines.

En claves secundarias indican que el registro actual no está relacionado con ninguno. En otros atributos indica que la tupla en cuestión carece de dicho atributo: por ejemplo en una tabla de *personas* un valor nulo en el atributo *teléfono* indicaría que dicha persona no tiene teléfono.

Es importante indicar que el texto vacío **‘ ’**, no significa lo mismo en un texto que el nulo; como tampoco el valor cero significa nulo.

Puesto que ese valor se utiliza continuamente, resulta imprescindible saber cómo actúa cuando se emplean operaciones lógicas sobre ese valor. Eso significa definir un tercer valor en la lógica booleana, además de los clásicos **verdadero** y **falso**. Un valor nulo no es ni verdadero ni falso (se suele interpretar como un *quizás*, o usando la aritmética clásica en valores lógicos, el 1 es verdadero, el 0 falso y el 0,5 nulo).

El uso de operadores lógicos con el nulo da lugar a que:

- **verdadero Y (AND) nulo** da como resultado, nulo
  (siguiendo la aritmética planteada antes: *1·0,5=0,5*)
- **falso Y (AND) nulo** da como resultado, falso (*0·0,5=0*)
- **verdadero O (OR) nulo** da como resultado, verdadero (*1+0,5>1*)
- **falso O nulo** da como resultado nulo (*0+0,5=0,5*)
- **la negación de nulo,** da como resultado nulo

Se utiliza un operador en todas las bases relacionales llamado **es nulo** *(is null)* que devuelve verdadero si el valor con el que se compara es nulo.

(2.3) restricciones

Se trata condiciones de obligado cumplimiento por las tuplas de la base de datos. Las hay de varios tipos.

(2.3.1) inherentes

Son aquellas que no son determinadas por los usuarios, sino que son definidas por el hecho de que la base de datos sea relacional. Las más importantes son:

- **No puede haber dos tuplas iguales**
- **El orden de las tuplas no es significativo**
- **El orden de los atributos no es significativo**
- **Cada atributo sólo puede tomar un valor en el dominio en el que está inscrito**

(2.3.2) semánticas

El modelo relacional permite a los usuario incorporar restricciones personales a los datos. Se comentan las diferentes reglas semánticas a continuación:

clave principal (primary key)

También llamada clave primaria. Marca uno o más atributos como identificadores de la tabla. De esa forma en esos atributos las filas de la tabla no podrán repetir valores ni tampoco dejarlos vacíos.

unicidad (unique)

Impide que los valores de los atributos marcados de esa forma, puedan repetirse. Esta restricción debe indicarse en todas las claves alternativas.

Al marcar una clave primaria se añade automáticamente sobre los atributos que forman la clave un criterio de unicidad.

obligatoriedad (not null)

Prohíbe que el atributo marcado de esta forma quede vacío (es decir impide que pueda contener el valor nulo, **null**).

integridad referencial (foreign key)

Sirve para indicar una clave externa (también llamada secundaria y foránea) sobre uno o más atributos. Los atributos marcados de esta forma sólo podrán contener valores que estén relacionados con la clave principal de la tabla que relacionan (llamada tabla principal). Dichos atributos sí podrán contener valores nulos.

Es decir si hay una tabla de alquileres en la que cada fila es un *alquiler*, existirá un atributo *cod_cliente* que indicará el *código del cliente* y que estará relacionado con una tabla de *clientes*, en la que dicho atributo es la clave principal. De hecho no se podrá incluir un código que no esté en la tabla clientes; eso es lo que prohíbe la integridad referencial.

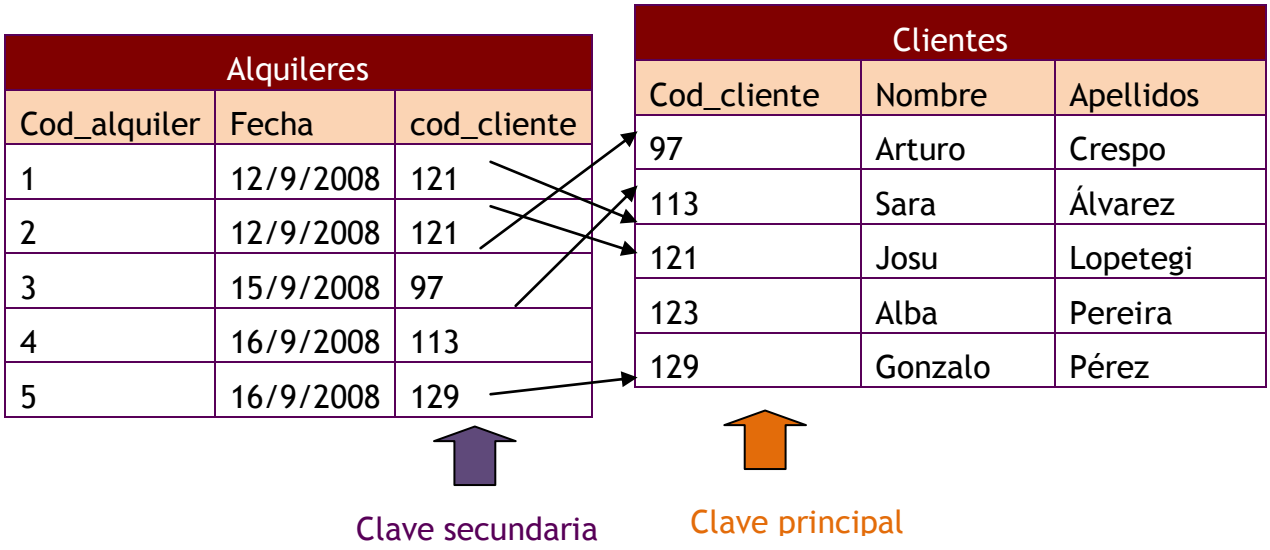


Ilustración 1, Ejemplo de clave secundaria

Eso causa problemas en las operaciones de borrado y modificación de registros; ya que si se ejecutan esas operaciones sobre la tabla principal (si se modifica o borra un cliente) quedarán filas en la tabla secundaria con la clave externa haciendo referencia a un valor que ya no existe en la tabla principal.

Para solventar esta situación se puede hacer uso de estas opciones:

- **Prohibir la operación** (*no action*).
- **Transmitir la operación en cascada** (*cascade*). Es decir si se modifica o borra un cliente; también se modificarán o barrarán los alquileres relacionados con él.
- **Colocar nulos** (*set null*) Las referencias al cliente en la tabla de alquileres se colocan como nulos (es decir, alquileres sin cliente).

- **Usar el valor por defecto *(default)*.** Se colocan un valor por defecto en las claves externas relacionadas. Este valor se indica al crear la tabla (opción **default**).

regla de validación (check)

Condición lógica que debe de cumplir un dato concreto para darlo por válido. Por ejemplo restringir el campo sueldo para que siempre sea mayor de 1000, sería una regla de validación. También por ejemplo que la fecha de inicio sea mayor que la fecha final.

disparadores o triggers

Se trata de pequeños programas grabados en la base de datos que se ejecutan automáticamente cuando se cumple una determinada condición. Sirven para realizar una serie de acciones cuando ocurre un determinado evento (cuando se añade una tupla, cuando se borra un dato, cuando un usuario abre una conexión…)

Los triggers permiten realizar restricciones muy potentes; pero son las más difíciles de crear.

(2.4) las 12 reglas de Codd

Preocupado por los productos que decían ser sistemas gestores de bases de datos relacionales (**RDBMS**) sin serlo, Codd publica las 12 reglas que debe cumplir todo DBMS para ser considerado relacional. Estas reglas en la práctica las cumplen pocos sistemas relacionales. Las reglas son:

(1) **Información**. Toda la información de la base de datos (**metadatos**) debe estar representada explícitamente en el esquema lógico. Es decir, **todos** los datos están en las tablas.

(2) **Acceso garantizado**. Todo **dato** es accesible sabiendo el valor de su clave y el nombre de la columna o atributo que contiene el dato.

(3) **Tratamiento sistemático de los valores nulos**. El DBMS debe permitir el tratamiento adecuado de estos valores. De ese modo el valor nulo se utiliza para representar la ausencia de información de un determinado registro en un atributo concreto.

(4) **Catálogo en línea basado en el modelo relacional.** Los metadatos deben de ser accesibles usando un esquema relacional. Es decir la forma de acceder a los metadatos es la misma que la de acceder a los datos.

(5) **Sublenguaje de datos completo**. Al menos debe de existir un lenguaje que permita el manejo completo de la base de datos. Este lenguaje, por lo tanto, debe permitir realizar cualquier operación sobre la misma.

(6) **Actualización de vistas.** El SGBD debe encargarse de que las vistas muestren la última información. No son válidas vistas que muestren datos que no están al día.

(7) **Inserciones, modificaciones y eliminaciones de dato nivel.** Cualquier operación de modificación debe actuar sobre conjuntos de filas o registros, nunca deben actuar registro a registro.

(8) **Independencia física.** Los datos deben de ser accesibles desde la lógica de la base de datos aún cuando se modifique el almacenamiento. La forma de acceder a los datos no varía porque el esquema físico de la base de datos, cambie.

(9) **Independencia lógica.** Los programas no deben verse afectados por cambios en las tablas. Que las tablas cambien no implica que cambien los programas.

(10) **Independencia de integridad.** Las reglas de integridad deben almacenarse en la base de datos (en el **diccionario de datos),** no en los programas de aplicación.

(11) **Independencia de la distribución.** El sublenguaje de datos debe permitir que sus instrucciones funciones igualmente en una base de datos distribuida que en una que no lo es.

(12) **No subversión**. Si el SGBD posee un lenguaje procedimental que permita crear bucles de recorrido fila a fila, éste no puede utilizarse para incumplir o evitar las reglas relacionales anteriores. Especialmente la regla 7 no puede ser incumplida por ningún lenguaje del SGBD.

(2.5) paso de entidad/relación al modelo relacional

(2.5.1) transformación de las entidades fuertes

En principio las entidades fuertes del modelo Entidad Relación son transformados al modelo relacional siguiendo estas instrucciones:

- **Entidades.** Las entidades pasan a ser tablas
- **Atributos**. Los atributos pasan a ser columnas o atributos de la tabla.
- **Identificadores principales**. Pasan a ser claves primarias
- **Identificadores candidatos**. Pasan a ser claves candidatas.

Esto hace que la transformación se produzca según este ejemplo:

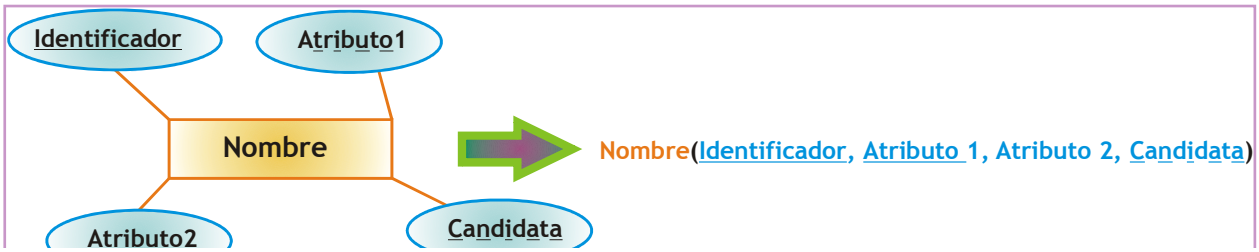


Ilustración 2,Transformación de una entidad fuerte al esquema relacional

(2.5.2) transformación de relaciones

La idea inicial es transformar cada relación del modelo conceptual en una tabla en el modelo relacional. Pero hay que tener en cuenta todos los casos

relaciones varios a varios

En las relaciones varios a varios (*n* a *n* en la cardinalidad mayor, la cardinalidad menor no cuenta para esta situación), la relación se transforma en una tabla cuyos atributos son: los atributos de la relación y las claves de las entidades relacionadas (que pasarán a ser claves externas). La clave de la tabla la forman todas las claves externas.

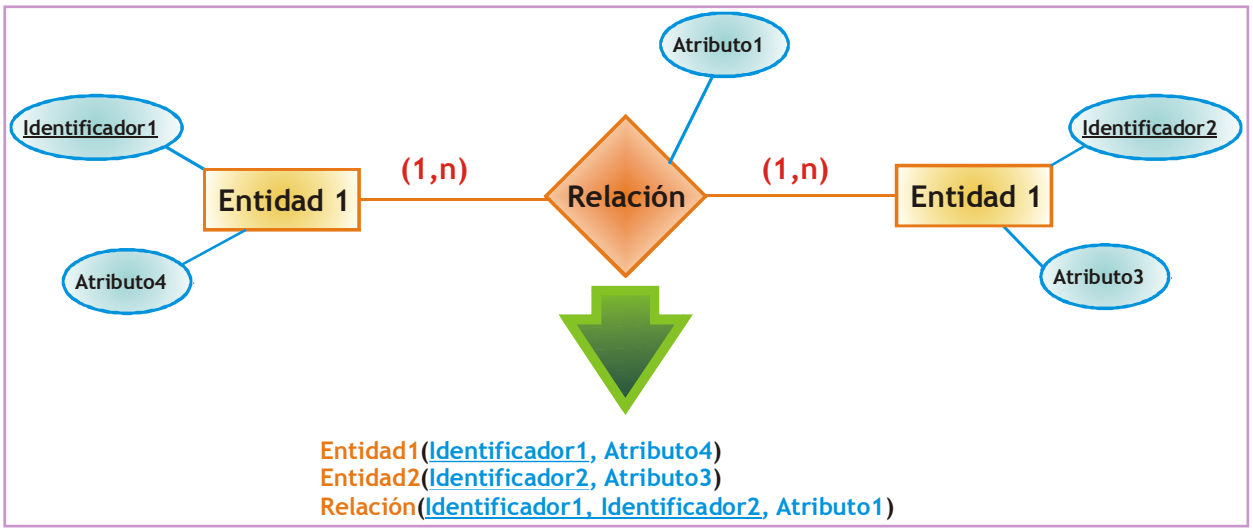


Ilustración 3, Transformación de una relación *n* a *n*

relaciones de orden *n*

Las relaciones ternarias, cuaternarias y ***n-arias*** que unen más de dos relaciones se transforman en una tabla que contiene los atributos de la relación más los identificadores de las entidades relacionadas. La clave la forman todas las claves externas:

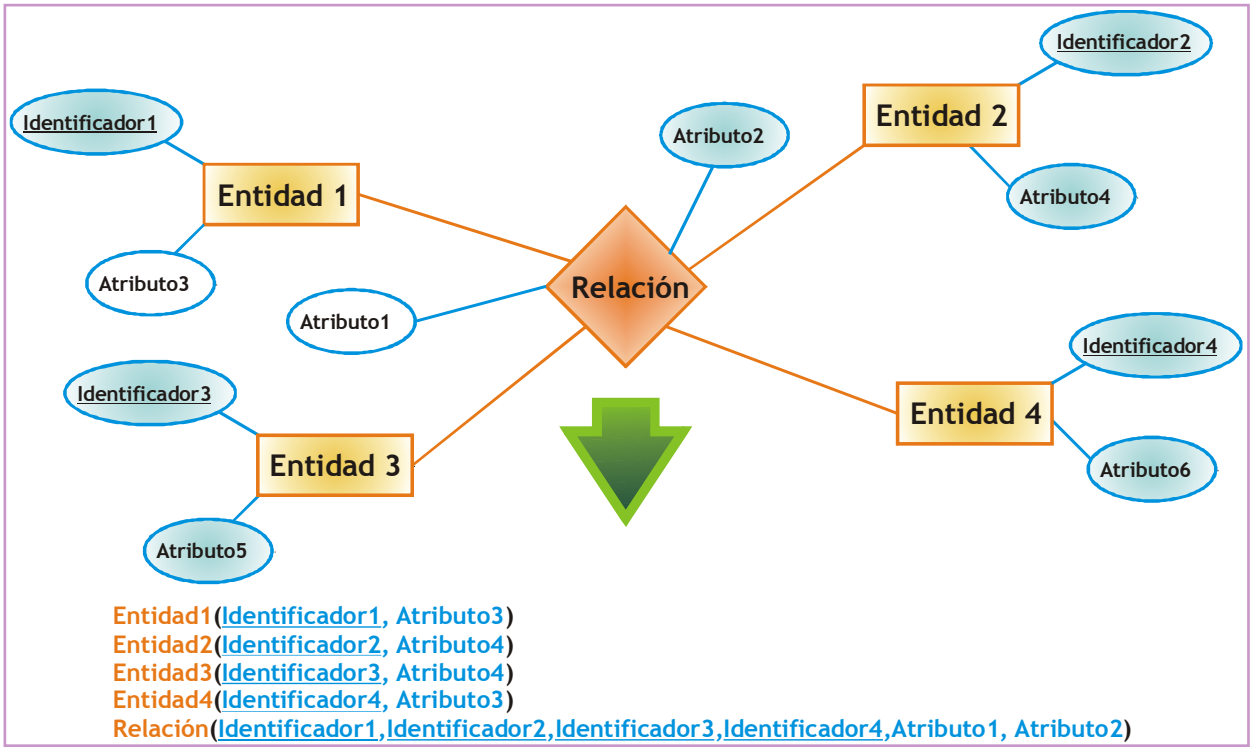


Ilustración 4, Transformación en el modelo relacional de una entidad n-aria

relaciones uno a varios

Las relaciones binarios de tipo uno a varios no requieren ser transformadas en una tabla en el modelo relacional. En su lugar la tabla del lado ***varios*** (**tabla** relacionada) incluye como clave externa[1] el identificador de la entidad del lado ***uno*** (**tabla principal**).

[1] **Clave externa, clave ajena, clave foránea, clave secundaria** y ***foreign key*** son sinónimos

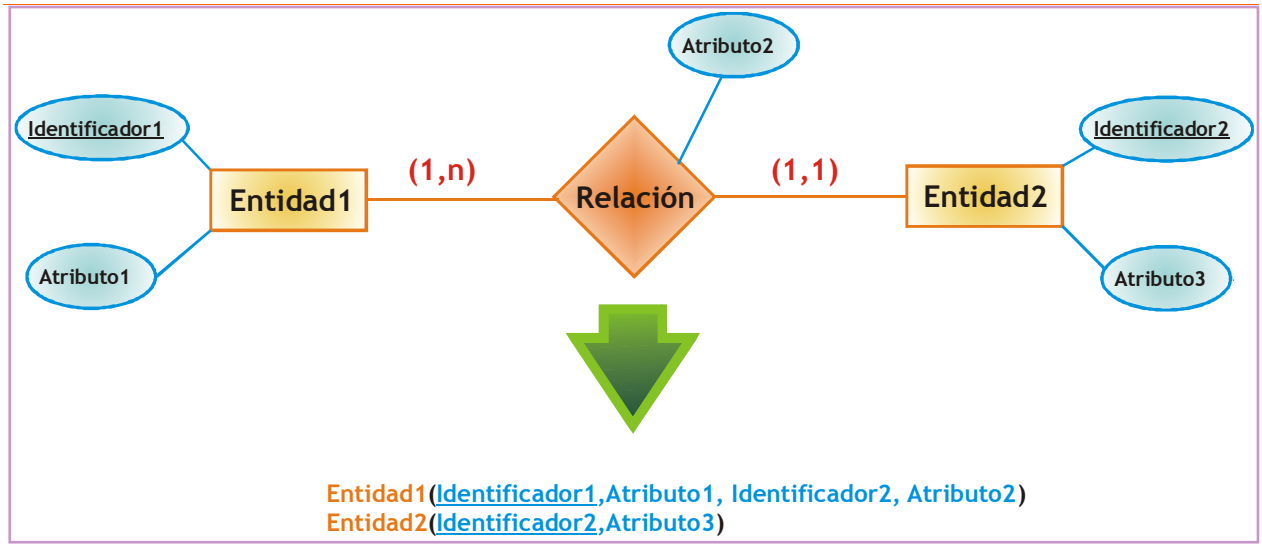


Ilustración 5, Transformación de una relación uno a varios

Así en el dibujo, el *Identificador2* en la tabla *Entidad1* pasa a ser una clave secundaria. En el caso de que el número mínimo de la relación sea de *cero* (puede haber ejemplares de la entidad uno sin relacionar), se deberá permitir valores nulos en la clave secundaria (en el ejemplo sería el *identificador2* en la *Entidad1*). En otro caso no se podrán permitir (ya que siempre habrá un valor relacionado).

relaciones 1 a 1

En el caso de las relaciones entre dos entidades con todas las cardinalidades a 1; hay dos posibilidades:

- Colocar la clave de una de las entidades como clave externa de la otra tabla (da igual cuál), teniendo en cuenta que dicha clave será **clave alternativa** además de ser clave secundaria.
- Generar una única tabla con todos los atributos de ambas entidades colocando como **clave principal** cualquiera de las claves de las dos entidades. La otra clave será marcada como **clave alternativa**. El nombre de la tabla sería el de la entidad más importante desde el punto de vista conceptual.

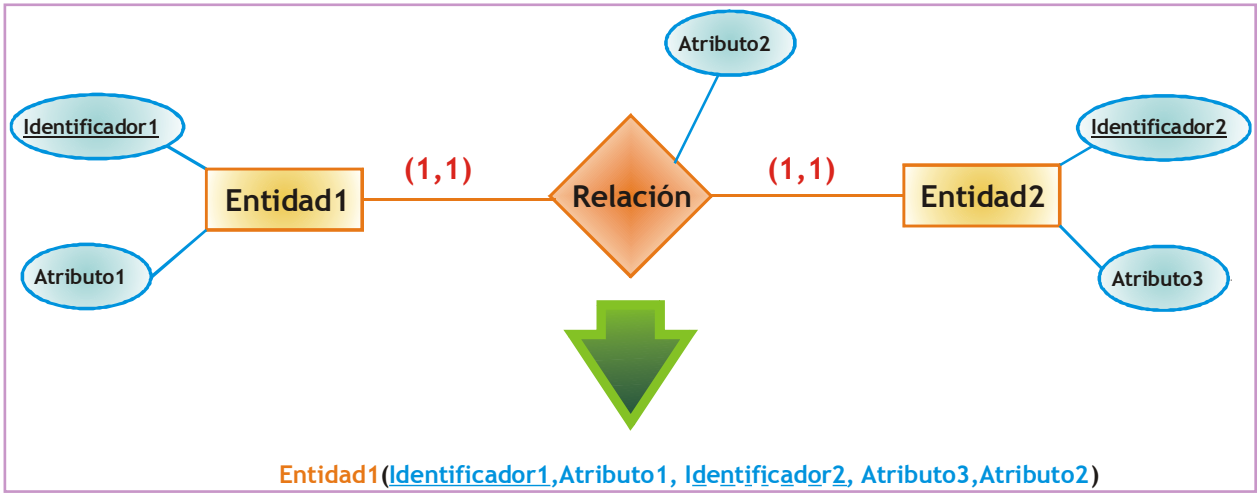


Ilustración 6, Posible solución a la cardinalidad 1 a 1

relaciones 0 a 1

Se trata de relaciones entre dos entidades con cardinalidad máxima de 1 en ambas direcciones, pero en una de ellas la cardinalidad mínima es 0. En este caso la solución difiere respecto a la anterior solución. No conviene generar una única tabla ya que habría numerosos valores nulos en la tabla (debido a que hay ejemplares que no se relacionan en las dos tablas).

La solución sería generar dos tablas, una para cada entidad. En la tabla con cardinalidad 0, se coloca como clave secundaria, la clave principal de la otra (dicha clave sería clave alternativa de esa tabla):

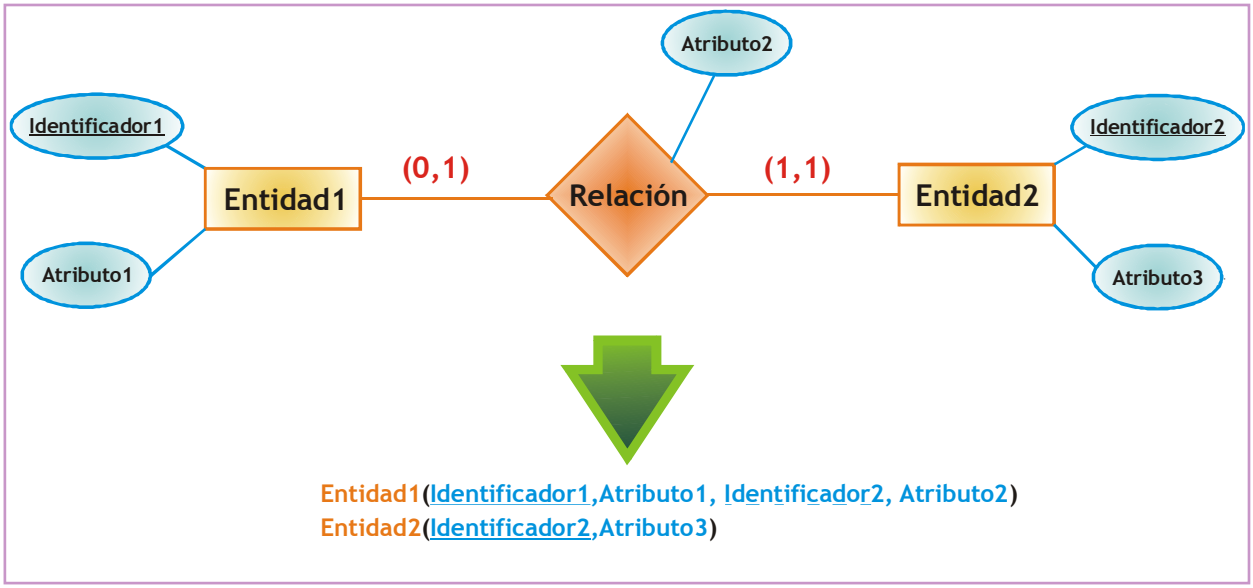


Ilustración 7, Solución a la relación 0 a 1

En el caso de que en ambos extremos nos encontremos con relaciones 0 a 1, entonces la solución es la misma, pero la clave que se copia en la tabla para ser clave secundaria, debe de ser tomada de la entidad que se relacione más con la

otra (la que esté más cerca de tener la cardinalidad 1 a 1 en el otro extremo). Dicha clave secundaria, en este caso, no será clave alternativa (pero sí tendría restricción de unicidad).

relaciones recursivas

Las relaciones recursivas se tratan de la misma forma que las otras, sólo que un mismo atributo puede figurar dos veces en una tabla como resultado de la transformación (por eso es interesante indicar el rol en el nombre del atributo.

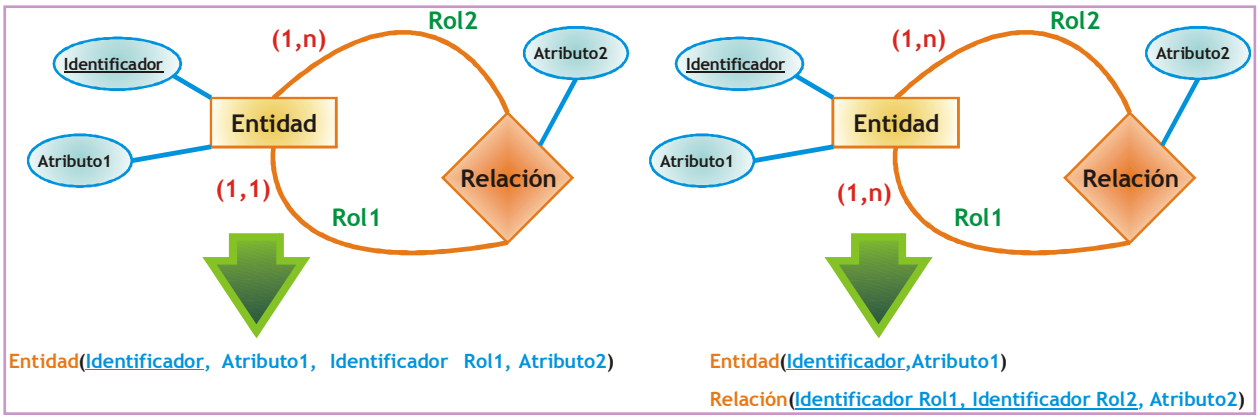


Ilustración 8, Transformación de relaciones recursivas en el modelo relacional

(2.5.3) entidades débiles

Toda entidad débil incorpora una relación implícita con una entidad fuerte. Esta relación no necesita incorporarse como tabla en el modelo relacional (al tratarse de una relación *n* a *1*), bastará con añadir como atributo y clave foránea en la entidad débil, el identificador de la entidad fuerte.

En ocasiones el identificador de la entidad débil tiene como parte de su identificador al identificador de la entidad fuerte (por ejemplo si para identificar líneas de factura utilizamos el *número de línea* y *el número de factura*, clave de la entidad *factura*). En esos casos no hace falta añadir de nuevo como clave externa el identificador de la entidad fuerte (imagen de la derecha)

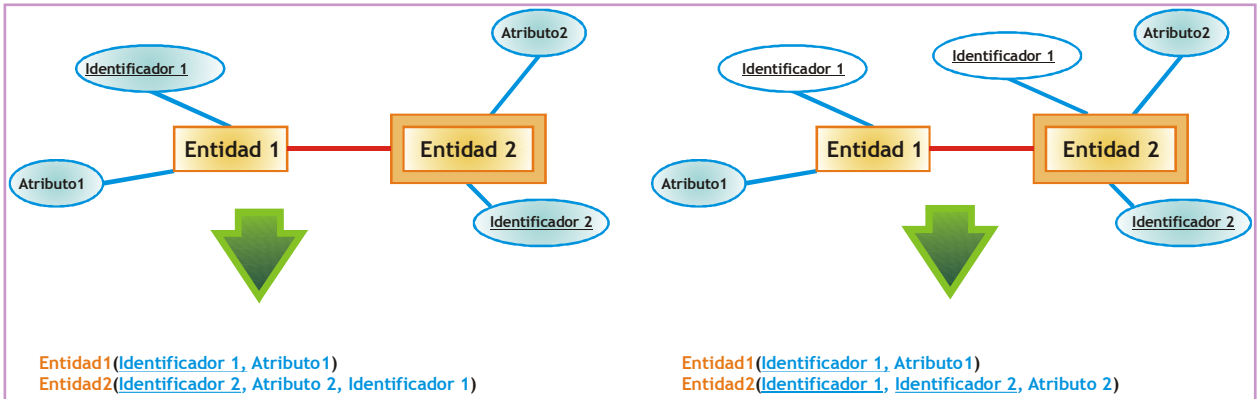


Ilustración 9, transformación de entidades débiles en el modelo relacional

(2.5.4) relaciones ISA

En el caso de las relaciones ISA, se siguen estas normas:

(1) Tanto las superentidades como las subentidades generarán tablas en el modelo relacional (en el caso de que la ISA sea de tipo total, se podría incluso no hacer la superentidad y pasar todos sus atributos a las subentidades, pero no es recomendable porque puede complicar enormemente el esquema interno).

(2) Los atributos se colocan en la tabla a la que se refiere a la entidad correspondiente

(3) En el caso de que las subentidades no hereden el identificador con la superentidad, se colocará en las subentidades el identificador de la superentidad como clave secundaria. Si además la relación ISA es de tipo **total**, entonces la clave secundaria además será clave alternativa.

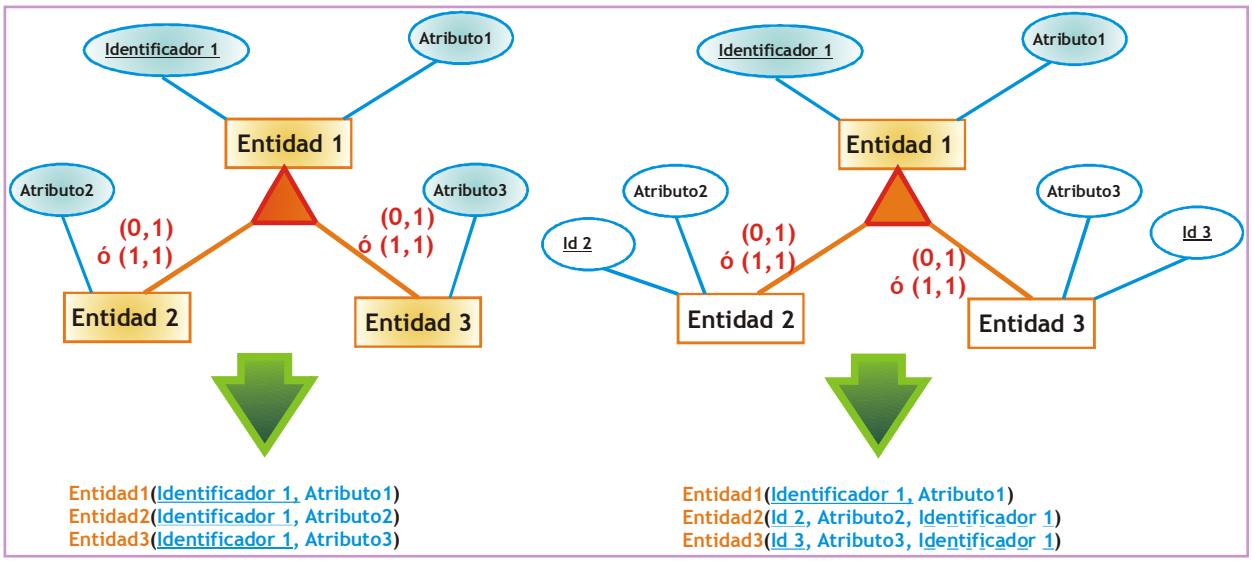


Ilustración 10, Proceso de transformación de relaciones ISA. A la izquierda cuando el identificador es heredado por las subentidades; a la derecha las subentidades tienen identificador propio

(4) Si la ISA es exclusiva o no, no cambia el esquema relacional, pero sí habrá que tenerlo en cuenta para las restricciones futuras en el esquema interno (casi siempre se realizan mediante triggers), ya que en las exclusivas no se puede repetir la clave de la superentidad en las subentidades.

(2.5.5) notas finales

El modelo conceptual entidad/relación es el verdadero mapa de la base de datos. Hay aspectos que no se reflejan al instante, por ejemplo el hecho de si la cardinalidad mínima es 0 o uno, o la obligatoriedad en una relación,.... Especial cuidado hay que tener con las relaciones ISA. Son aspectos a tener en cuenta en el siguiente modelo (en el interno) al crear por ejemplo índices y restricciones.

Por ello ese modelo es la referencia obligada de los profesionales de la base de datos (en especial de los administradores) y su contenido no debe dejar de tenerse en cuenta aunque ya tengamos el esquema relacional.

(2.6) representación de esquemas de bases de datos relacionales

En el tema 3, ya vimos como eran los esquemas relacionales. Ejemplo:

PIEZA(Tipo, Modelo, Nombre, Apellido1, Apellido2)

EMPRESA(CIF, Cod_Empresa, Nombre, Dirección)

SUMINISTROS(Tipo,Modelo, Cod_Empresa, Precio)

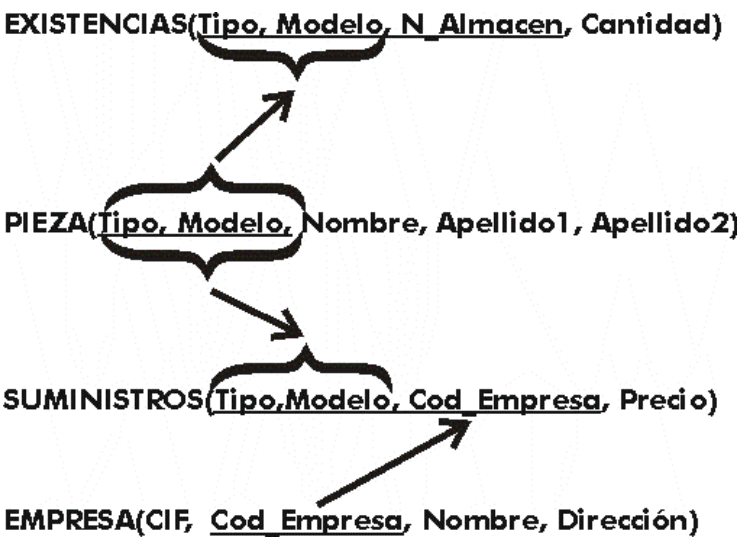
EXISTENCIAS(Tipo, Modelo, N_Almacen, Cantidad)

En ese tipo de esquemas es difícil ver las relaciones en los datos, algo que sí se ve muy bien en los esquemas entidad relación. Por ello se suelen complementar los esquemas clásicos con líneas y diagramas que representan esa información.

(2.6.1) Grafos relacionales

Es un esquema relacional en el que hay líneas que enlazan las claves principales con las claves secundarias para representar mejor las relaciones. A veces se representa en forma de nodos de grafos y otras se complementa el clásico.

Ejemplo:

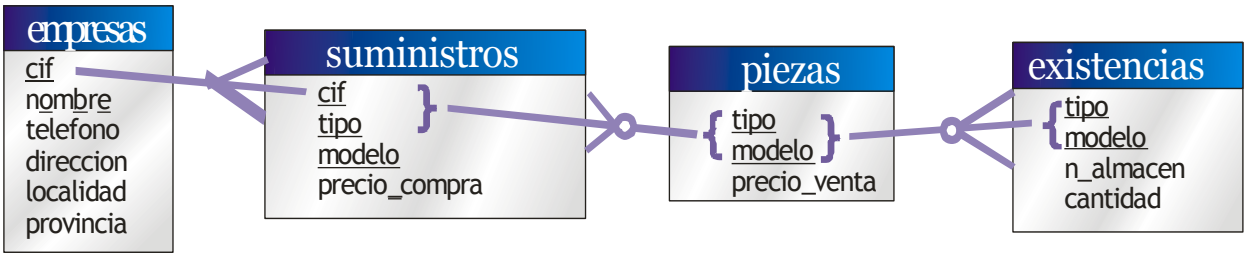


(2.6.2) Esquemas relacionales derivados del modelo entidad/relación

Hay quien los llama **esquemas entidad/relación relacionales**. De hecho es una mezcla entre los esquemas relacionales y los entidad/relación. Hoy en día se utiliza mucho, en especial por las herramientas CASE de creación de diseños de bases de datos.

Las tablas se representan en forma de rectángulo que contiene una fila por cada atributo y una fila inicial para la cabecera en la que aparece el nombre de la tabla. Después aparecen líneas que muestran la relación entre las claves y su cardinalidad.

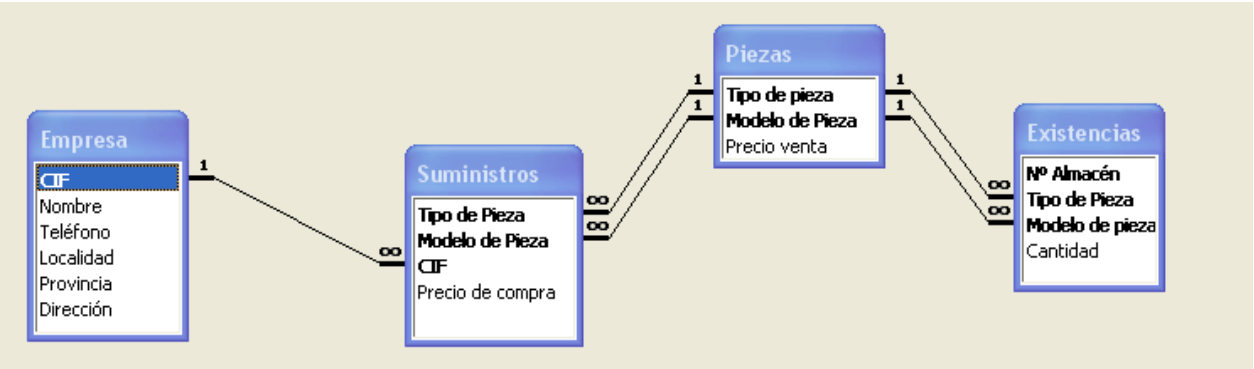
Uno de los más utilizados actualmente es éste:



Las cardinalidades se pueden mostrar en otros formatos, pero siempre se mostrarán en este tipo de esquemas. En este caso el inicio de la línea (en la clave principal) se considera cardinalidad 1 y en el extremo podemos tener un final de línea sin símbolos (cardinalidad 1,1), acabado en varias ramas (cardinalidad 1,n) o con un círculo (cardinalidad mínima de 0)

Se ha hecho muy popular la forma de presentar esquemas relacionales del programa Microsoft Access.

Ejemplo:



Es otra forma muy clara de representar relaciones y cardinalidades (aunque tiene problemas para representar relaciones de dos o más atributos).
Sin duda los esquemas más completos son los que reflejan no sólo las cardinalidades sino también todas las restricciones (e incluso los tipos de datos, aunque esto ya es una competencia del esquema interno). Véase el esquema de la Ilustración 11. En ese esquema los símbolos funcionan de esta forma:

Símbolo	Ejemplo	Significado
Subrayado	**DNI**	Clave principal
Subrayado discontinuo	Clave2	Clave alternativa
º	Nombreº	No admite valores nulos (restricción *NOT NULL*)
*	Nombre*	No admite duplicados (restricción *UNIQUE*)

Además los campos que están el final de una flecha son claves secundarias.

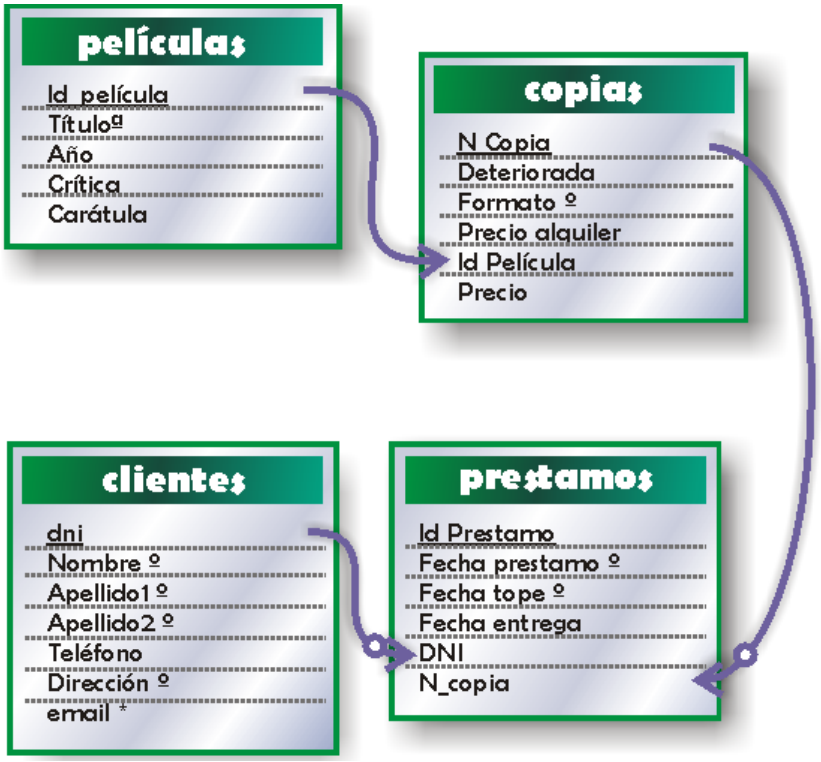


Ilustración 11, Esquema relacional completo de la base de datos de un Video Club

El programa Visio de Microsoft (y algunos otros más), representan las restricciones con letras:

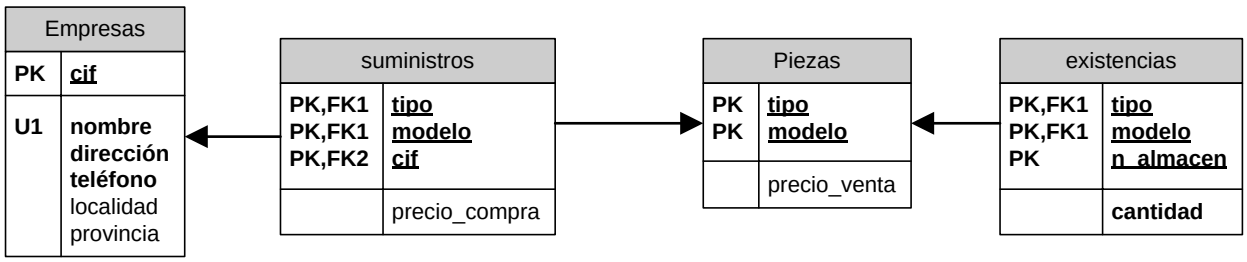


Ilustración 12, Esquema relacional del almacén según el programa Visio de Microsoft

En este caso los símbolos **PK** significan ***Primary Key*** (clave principal), **FK** es ***Foreign Key*** (clave secundaria, los números sirven para distinguir unas claves de otras) y **UK** es ***Unique*** (unicidad).

(2.7) normalización

(2.7.1) problemas del esquema relacional

Una vez obtenido el esquema relacional resultante del esquema entidad/relación que representa la base de datos, normalmente tendremos una buena base de datos. Pero otras veces, debido a fallos en el diseño o a problemas indetectables, tendremos un esquema que puede producir una base de datos que incorpore estos problemas:

- **Redundancia.** Se llama así a los datos que se repiten continua e innecesariamente por las tablas de las bases de datos. Cuando es excesiva es evidente que el diseño hay que revisarlo, es el primer síntoma de problemas y se detecta fácilmente.
- **Ambigüedades.** Datos que no clarifican suficientemente el registro al que representan. Los datos de cada registro podrían referirse a más de un registro o incluso puede ser imposible saber a qué ejemplar exactamente se están refiriendo. Es un problema muy grave y difícil de detectar.
- **Pérdida de restricciones de integridad.** Normalmente debido a **dependencias funcionales.** Más adelante se explica este problema. Se arreglan fácilmente siguiendo una serie de pasos concretos.
- **Anomalías en operaciones de modificación de datos.** El hecho de que al insertar un solo elemento haya que repetir tuplas en una tabla para variar unos pocos datos. O que eliminar un elemento suponga eliminar varias tuplas necesariamente (por ejemplo que eliminar un cliente suponga borrar seis o siete filas de la tabla de clientes, sería un error muy grave y por lo tanto un diseño terrible).

El principio fundamental reside en que las tablas deben referirse a objetos o situaciones muy concretas, relacionados exactamente con elementos reconocibles por el sistema de información de forma inequívoca. Cada fila de una tabla representa inequívocamente un elemento reconocible en el sistema. Lo que ocurre es que conceptualmente es difícil agrupar esos elementos correctamente.

En cualquier caso la mayor parte de problemas se agravan si no se sigue un modelo conceptual y se decide crear directamente el esquema relacional. En ese caso el diseño tiene una garantía casi asegurada de funcionar mal.

Cuando aparecen los problemas enumerados entonces se les puede resolver usando reglas de normalización. Estas reglas suelen forzar la división de una tabla en dos o más tablas para arreglar ese problema.

(2.7.2) formas normales

Las formas normales se corresponde a una teoría de normalización iniciada por el propio **Codd** y continuada por otros autores (entre los que destacan **Boyce** y **Fagin**). Codd definió en 1970 la primera forma normal, desde ese momento aparecieron la segunda, tercera, la Boyce-Codd, la cuarta y la quinta forma normal.

Una tabla puede encontrarse en primera forma normal y no en segunda forma normal, pero no al contrario. Es decir los números altos de formas normales son más restrictivos (la quinta forma normal cumple todas las anteriores).

La teoría de formas normales es una teoría absolutamente matemática, pero en el presente manual se describen de forma más intuitiva.

Hay que tener en cuenta que muchos diseñadores opinan que basta con llegar a la forma Boyce-Codd, ya que la cuarta, y sobre todo la quinta, forma normal es polémica. Hay quien opina que hay bases de datos peores en quinta forma normal que en tercera. En cualquier caso debería ser obligatorio para cualquier diseñador llegar hasta la forma normal de Boyce-Codd.

(2.7.3) primera forma normal (1FN)

Es una forma normal inherente al esquema relacional. Es decir toda tabla realmente relacional la cumple.

Se dice que una tabla se encuentra en primera forma normal si impide que un atributo de una tupla pueda tomar más de un valor. La tabla:

TRABAJADOR		
DNI	**Nombre**	**Departamento**
12121212A	Andrés	Mantenimiento
12345345G	Andrea	Dirección Gestión

Visualmente es un tabla, pero no una tabla relacional (lo que en terminología de bases de datos relacionales se llama **relación**). No cumple la primera forma normal.

Sería primera forma normal si los datos fueran:

TRABAJADOR		
DNI	**Nombre**	**Departamento**
12121212A	Andrés	Mantenimiento
12345345G	Andrea	Dirección
12345345G	Andrea	Gestión

Esa tabla sí esta en primera forma normal.

(2.7.4) dependencias funcionales

dependencia funcional

Se dice que un conjunto de atributos (Y) depende funcionalmente de otro conjunto de atributos (X) si para cada valor de X hay un único valor posible para Y. Simbólicamente se denota por X→Y.

Por ejemplo el *nombre* de una persona depende funcionalmente del *DNI*; es decir para un DNI concreto sólo hay un nombre posible. En la tabla del ejemplo anterior, el departamento no tiene dependencia funcional, ya que para un mismo DNI puede haber más de un departamento posible. Pero el nombre sí que depende del DNI.

Al conjunto X del que depende funcionalmente el conjunto Y se le llama **determinante**. Al conjunto Y se le llama **implicado.**

dependencia funcional completa

Un conjunto de atributos (Y) tiene una dependencia funcional completa sobre otro conjunto de atributos (X) si Y tiene dependencia funcional de X y además no se puede obtener de X un conjunto de atributos más pequeño que consiga una dependencia funcional de Y (es decir, no hay en X un determinante formado por atributos más pequeños).

Por ejemplo en una tabla de clientes, el conjunto de atributos formado por el *nombre* y el *dni* producen una dependencia funcional sobre el atributo *apellidos*. Pero no es plena ya que el *dni* individualmente, también produce una dependencia funcional sobre *apellidos*. El dni sí produce una dependencia funcional completa sobre el campo apellidos.

Una dependencia funcional completa se denota como X⇒Y

dependencia funcional elemental

Se produce cuando X e Y forman una dependencia funcional completa y además Y es un único atributo.

dependencia funcional transitiva

Es más compleja de explicar, pero tiene también utilidad. Se produce cuando tenemos tres conjuntos de atributos X, Y y Z. Y depende funcionalmente de X (X→Y), Z depende funcionalmente de Y (Y→Z). Además X no depende funcionalmente de Y (Y-/→X). Entonces ocurre que X produce una dependencia funcional transitiva sobre Z.

Esto se denota como: (X—→Z)

Por ejemplo si X es el atributo *Número de Clase* de un instituto, e Y es el atributo *Código Tutor*. Entonces X→Y (el tutor depende funcionalmente del número de clase). Si Z representa el *Código del departamento*, entonces Y→Z (el código del departamento depende funcionalmente del código tutor, cada tutor sólo puede estar en un departamento). Como ocurre que Y-/→X (el código de la clase no depende funcionalmente del código tutor, un código tutor se

puede corresponder con varios códigos de clase). Entonces X—→Z (el código del departamento depende transitivamente del código de la clase).

(2.7.5) segunda forma normal (2FN)

Ocurre si una tabla está en primera forma normal y además cada atributo que no sea clave, depende de forma funcional completa respecto de cualquiera de las claves. Toda la clave principal debe hacer dependientes al resto de atributos, si hay atributos que depende sólo de parte de la clave, entonces esa parte de la clave y esos atributos formarán otra tabla. Ejemplo:

ALUMNOS				
DNI	**Cod Curso**	**Nombre**	**Apellido1**	**Nota**
12121219A	34	Pedro	Valiente	9
12121219A	25	Pedro	Valiente	8
3457775G	34	Ana	Fernández	6
5674378J	25	Sara	Crespo	7
5674378J	34	Sara	Crespo	6

Suponiendo que el DNI y el código de curso formen una clave principal para esta tabla, sólo la nota tiene dependencia funcional completa. El nombre y los apellidos dependen de forma completa del DNI. La tabla no es 2FN, para arreglarlo:

ALUMNOS		
DNI	**Nombre**	**Apellido1**
12121219A	Pedro	Valiente
3457775G	Ana	Fernández
5674378J	Sara	Crespo

ASISTENCIA		
DNI	**Cod Curso**	**Nota**
12121219A	34	9
12121219A	25	8
3457775G	34	6
5674378J	25	7
5674378J	34	6

(2.7.6) tercera forma normal (3FN)

Ocurre cuando una tabla está en 2FN y además ningún atributo que no sea clave depende transitivamente de las claves de la tabla. Es decir no ocurre cuando algún atributo depende funcionalmente de atributos que no son clave.

Ejemplo:

ALUMNOS				
DNI	**Nombre**	**Apellido1**	**Cod Provincia**	**Provincia**
12121349A	Salvador	Velasco	34	Palencia
12121219A	Pedro	Valiente	34	Palencia
3457775G	Ana	Fernández	47	Valladolid
5674378J	Sara	Crespo	47	Valladolid
3456858S	Marina	Serrat	08	Barcelona

La Provincia depende funcionalmente del código de provincia, lo que hace que no esté en 3FN. El arreglo sería:

ALUMNOS			
DNI	**Nombre**	**Apellido1**	**Cod Provincia**
12121349A	Salvador	Velasco	34
12121219A	Pedro	Valiente	34
3457775G	Ana	Fernández	47
5674378J	Sara	Crespo	47
3456858S	Marina	Serrat	08

PROVINCIA	
Cod Provincia	**Provincia**
34	Palencia
47	Valladolid
08	Barcelona

(2.7.7) forma normal de Boyce-Codd (FNBC o BCFN)

Ocurre si una tabla está en tercera forma normal y además todo determinante es una clave candidata. Ejemplo:

TUTORÍAS		
DNI	**Asignatura**	**Tutor**
12121219A	Lenguaje	Eva
12121219A	Matemáticas	Andrés
3457775G	Lenguaje	Eva
5674378J	Matemáticas	Guillermo
5674378J	Lenguaje	Julia
5634823H	Matemáticas	Guillermo

Esa tabla está en tercera forma normal (no hay dependencias transitivas), pero no en forma de Boyce - Codd, ya que *(DNI, Asignatura) →Tutor* y *Tutor→Asignatura* y *Tutor-/→(DNI,Asignatura)*. En este caso la redundancia ocurre por mala selección de clave. La redundancia de la asignatura es completamente evitable. La solución sería:

TUTORÍAS	
DNI	**Tutor**
12121219A	Eva
12121219A	Andrés
3457775G	Eva
5674378J	Guillermo
5674378J	Julia
5634823H	Guillermo

ASIGNATURASTUTOR	
Asignatura	**Tutor**
Lenguaje	Eva
Matemáticas	Andrés
Matemáticas	Guillermo
Lenguaje	Julia

En las formas de Boyce-Codd hay que tener cuidado al descomponer ya que se podría perder información por una mala descomposición

(2.7.8) cuarta forma normal (4FN). dependencias multivaluadas

dependencia multivaluada

Para el resto de formas normales (las diseñadas por Fagin, mucho más complejas), es importante definir este tipo de dependencia, que es distinta de las funcionales. Si las funcionales eran la base de la segunda y tercera forma normal (y de la de Boyce-Codd), éstas son la base de la cuarta forma normal.

Una dependencia multivaluada de X sobre Y (es decir X->>Y), siendo X e Y atributos de la misma tabla, ocurre cuando Y tiene un conjunto de valores bien definidos sobre cualquier valor de X. Es decir, dado X sabremos los posibles valores que puede tomar Y.

Se refiere a posibles valores (en plural) y se trata de que los valores de ese atributo siempre son los mismos según el valor de un atributo y no del otro.

Ejemplo:

Nº Curso	Profesor	Material
17	Eva	1
17	Eva	2
17	Julia	1
17	Julia	2
25	Eva	1
25	Eva	2
25	Eva	3

La tabla cursos, profesores y materiales del curso. La tabla está en FNBC ya que no hay dependencias transitivas y todos los atributos son clave sin dependencia funcional hacia ellos. Sin embargo hay redundancia. Los materiales se van a repetir para cualquier profesor dando cualquier curso, ya que los profesores van a utilizar todos los materiales del curso (de no ser así no habría ninguna redundancia).

Los materiales del curso dependen de forma multivaluada del curso y no del profesor en una dependencia multivaluada (no hay dependencia funcional ya que los posibles valores son varios). Para el par *Nº de curso* y *Profesor* podemos saber los materiales; pero lo sabemos por el curso y no por el profesor.

cuarta forma normal

Ocurre esta forma normal cuando una tabla está en forma normal de Boyce Codd y toda dependencia multivaluada es una dependencia funcional. Para la tabla anterior la solución serían dos tablas:

Nº Curso	Material
17	1
17	2
25	1
25	2
25	3

Nº Curso	Profesor
17	Eva
17	Julia
25	Eva

Un teorema de Fagin indica cuando hay tres pares de conjuntos de atributos **X, Y** y **Z** si ocurre **X->>Y** y **X->>Z** (**Y** y **Z** tienen dependencia multivaluada sobre **X**), entonces las tablas **X, Y** y **X, Z** reproducen sin perder información lo que poseía la tabla original. Este teorema marca la forma de dividir las tablas hacia una 4FN

(2.7.9) quinta forma normal (5FN)

dependencias de JOIN o de reunión

Una proyección de una tabla es la tabla resultante de tomar un subconjunto de los atributos de una tabla (se trata de la operación proyección, Π, del álgebra relacional).

Se dice que se tiene una tabla con dependencia de tipo **JOIN** si se puede obtener esa tabla como resultado de combinar (mediante la operación JOIN del álgebra relacional) varias proyecciones de la misma.

quinta forma normal

Ocurre cuando está en 4FN y además no hay proyecciones que combinadas formen la tabla original, o si las hay son consecuencia de aplicar la clave principal. Es la más compleja y polémica de todas. Polémica pues no está claro en muchas ocasiones está muy claro que el paso a 5FN mejore la base de datos. Fue definida también por Fagin.

En definitiva una tabla está en 5FN si está en 4FN y no hay restricciones impuestas por el creador de la base de datos.

Es raro encontrarse este tipo de problemas cuando la normalización llega a 4FN. Se deben a restricciones muy concretas.

Ejemplo:

Proveedor	Material	Proyecto
1	1	2
1	2	1
2	1	1
1	1	1

Indican códigos de material suministrado por un proveedor y utilizado en un determinado proyecto.

Si ocurre una restricción especial como por ejemplo: Cuando un proveedor nos ha suministrado alguna vez un determinado material, si ese material aparece en otro proyecto, haremos que el proveedor nos suministre también ese material para ese proyecto.

Eso ocurre en los datos como el proveedor número 1 nos suministró el material número 1 para el proyecto 2 y en el proyecto 1 utilizamos el material 1, aparecerá la tupla proveedor 1, material 1 y proyecto 1.

La dependencia que produce esta restricción es lejana y se la llama **de reunión**. Para esa restricción esta división en tablas sería válida:

Proveedor	Material
1	1
1	2
2	1

Material	Proyecto
1	2
2	1
1	1

Esa descomposición no pierde valores en este caso, sabiendo que si el proveedor nos suministra un material podremos relacionarle con todos los proyectos que utilizan ese material.

Resumiendo, una tabla no está en quinta forma normal si hay una descomposición de esa tabla que muestre la misma información que la original.

Normalmente se crean tablas en quinta forma normal cuando en la misma tabla hay muchos atributos y es casi inmanejable o cuando hay muchos registros

y pocos atributos. En el caso de que haya muchos atributos se divide la tabla en dos donde la clave es la misma en ambas tablas.

(2.8) índice de ilustraciones