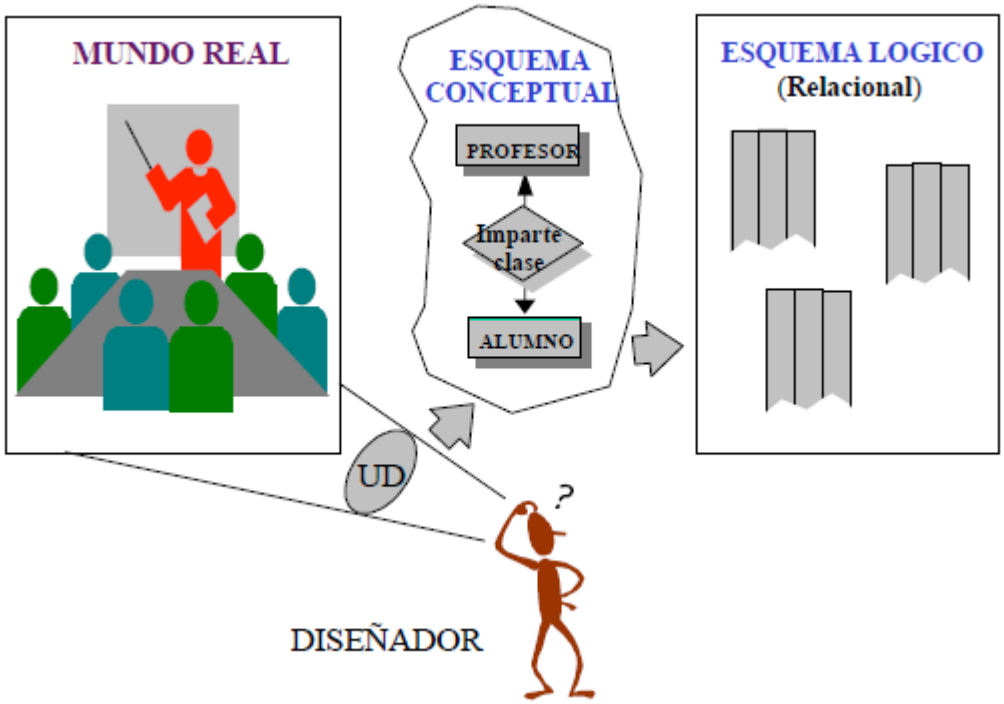
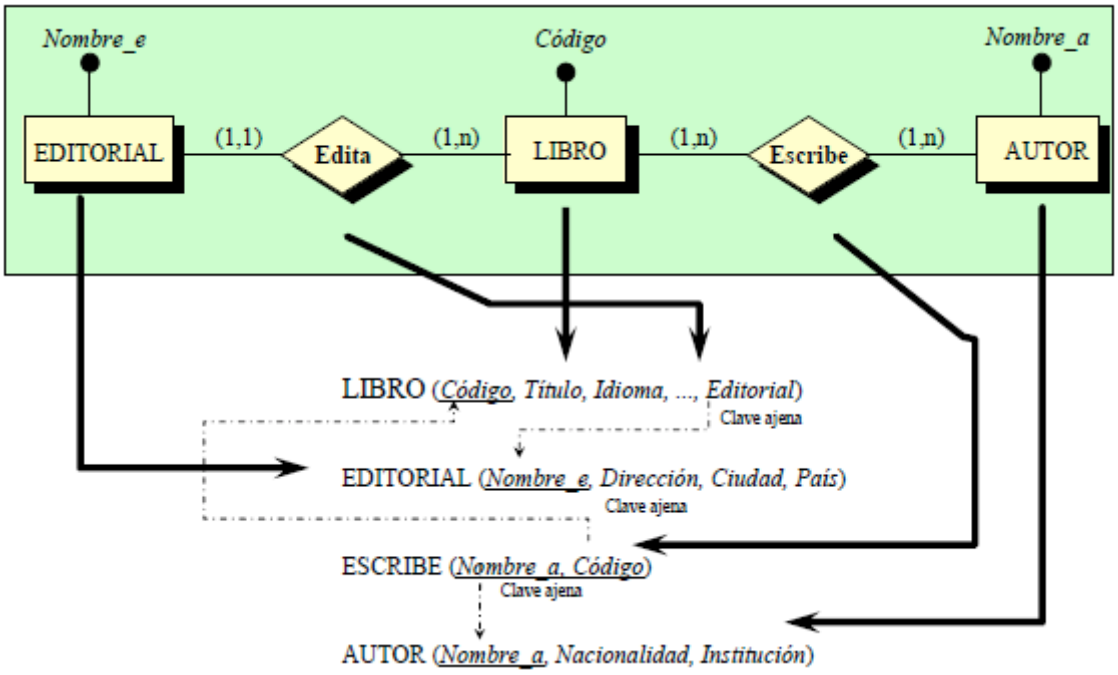
MODELO DE DATOS RELACIONAL




Nombre_e
Código
Nombre_a
EDITORIAL
(1,1)
Edita
(1,n)
LIBRO
(1,n)
Escribe
(1,n)
AUTOR
LIBRO (Código, Título, Idioma, ..., Editorial)
Clave ajena
EDITORIAL (Nombre_e, Dirección, Ciudad, País)
Clave ajena
ESCRIBE (Nombre_a, Código)
Clave ajena
AUTOR (Nombre_a, Nacionalidad, Institución)

1. HISTORIA Y OBJETIVOS

Cuando en el año 1970 el Dr. E. F. Codd propone un nuevo modelo de datos, los SGBD imperantes en el mercado, de tipo Codasyl y Jerárquico, no habían logrado superar el grave inconveniente que suponía la dependencia de las aplicaciones desarrolladas en ellos respecto a las estructuras de los datos.

A diferencia de estos modelos de datos basados en punteros físicos por los que tenía que navegar el programador a fin de recuperar y actualizar los datos, el Modelo Relacional (MR) se propone, como principal objetivo, aislar al usuario de las estructuras físicas de los datos, consiguiendo así la independencia de las aplicaciones respecto de los datos, finalidad perseguida desde los inicios de las bases de datos.

En el nuevo modelo, basado en la teoría matemática de las relaciones, los datos se estructuran lógicamente en forma de relaciones (tablas). Esta formalización matemática convirtió rápidamente al modelo en una fuente fundamental de la investigación en bases de datos.

Relación ~ tabla

Los avances más importantes que el modelo de datos relacional incorpora respecto a los modelos de datos anteriores son:

- **Sencillez y uniformidad:** Los usuarios ven la base de datos relacional como una colección de tablas, y al ser la tabla la estructura fundamental del modelo, éste goza de una gran uniformidad, lo que unido a unos lenguajes no navegacionales y muy orientados al usuario final, da como resultado la sencillez de los sistemas relacionales.
- **Sólida fundamentación teórica:** Al estar el modelo definido con rigor matemático, el diseño y la evaluación del mismo puede realizarse por métodos sistemáticos basados en abstracciones.
- **Independencia de la interfaz de usuario:** los lenguajes relacionales, al manipular conjuntos de registros, proporcionan una gran independencia respecto a la forma en la que los datos están almacenados.

Puesto que el objetivo de este capítulo es establecer los fundamentos del modelo relacional a fines de diseño, sólo nos interesa la parte estática del modelo y, dentro de ella, aquellos aspectos más relacionados con el diseño relacional de bases de datos, como pueden ser, por ejemplo, las restricciones.

2. ELEMENTOS PERMITIDOS

La estructura básica, y única, del modelo relacional es la relación (también llamada *tabla*), que sirve para representar tanto los objetos como las asociaciones entre ellos. En la tabla se distinguen un conjunto de columnas (*los atributos*) y un conjunto de filas (*tuplas*). Los *atributos* son las propiedades de las relaciones, y se definen sobre los *dominios*, los cuales, a diferencia de los atributos, tienen vida propia, es decir, existen con independencia de cualquier otro elemento del modelo, mientras que la existencia de un atributo va unida a la de la relación a la que pertenece.

atributo 1	atributo 2	.....	atributo n	
xxxxxxxx	xxxxxxxxx	.....	xxxxxxx	→ tupla 1
xxxxxxxx	xxxxxxxxx	.....	xxxxxxx	→ tupla 2
xxxxxxxx	xxxxxxxxx	.....	xxxxxxx	.
xxxxxxxx	xxxxxxxxx	.....	xxxxxxx	.
xxxxxxxx	xxxxxxxxx	.....	xxxxxxx	.
xxxxxxxx	xxxxxxxxx	.....	xxxxxxx	.
xxxxxxxx	xxxxxxxxx	.....	xxxxxxx	→ tupla n

RELACIÓN ALUMNOS

NUM_MAT	NOMBRE	APELLIDOS	CURSO	← ATRIBUTOS
5467	JUAN	CABELLO	1BACH-A	← TUPLAS
3421	DOLORES	GARCÍA	1BACH-C	← TUPLAS
7622	JESÚS	SÁNCHEZ	2BACH-C	← TUPLAS

Fig. 1. Ejemplos de representación de una relación en forma de tabla

2.1. Dominios, Relaciones y Atributos

El *Universo del Discurso* (UD) de una base de datos relacional está compuesto por un conjunto de dominios $\{D_i\}$ y de un conjunto de relaciones $\{R_i\}$ definidas sobre los dominios.

Un **dominio** (D) es un conjunto nominado, finito y homogéneo de valores atómicos. Cada dominio se especifica lógicamente mediante un nombre y un formato, el cual puede definirse por extensión (dando sus posibles valores) o por intensión (mediante un tipo de datos). A veces se asocia al dominio su unidad de medida (kilos, metros, etc.) y ciertas restricciones (como un rango de valores).

Por ejemplo, podemos definir el domino *Materias*, cuyo conjunto de valores, definido por extensión, podría ser: Bases de Datos, Sistemas Operativos, Lenguajes, etc. Otro dominio podría ser *Códigos*, definido por intensión como cadena de caracteres.

Un **atributo** (A) es la interpretación de un determinado dominio en una relación, es decir el "papel" que desempeña en la misma; si D es el dominio de A se denota:

$$D = Dom(A)$$

Por ejemplo, en la relación CURSO, un atributo puede ser *Cód_curso* y otro *Materia* definidos, respectivamente, sobre los dominios: *Códigos* y *Materias*.

$$Códigos = Dom(Cód\_curso)$$

$$Materias = Dom(Materia)$$

Un atributo y un dominio pueden llamarse igual, pero hay que tener en cuenta que:

- Un atributo está siempre asociado a una relación, mientras que un dominio tiene existencia propia con independencia de las relaciones.
- Un atributo representa una propiedad de una relación.
- Un atributo toma valores de un dominio.
- Varios atributos distintos (de la misma o de diferentes relaciones) pueden tomar sus valores del mismo dominio.

Matemáticamente, una relación definida sobre un conjunto de dominios $D_i...D_n$. (no necesariamente distintos) es un subconjunto del producto cartesiano (todas las posibles combinaciones distintas de valores) de los *n* dominios (donde *n* es el *grado de la relación*).

Nº Emple	Apellidos	Salario	Numdepart	FechaAlta
13407877B	Milagros Suela Sarro	1.500	10	18/11/90
41667891C	José María Cabello	2.000	20	29/10/92

Esquema = cabecera = estructura = intensión

Relación = cuerpo = tuplas = extensión

Fig. 2. Ejemplos de grado de la relación, en este caso 5

Podemos precisar mejor el concepto de relación si lo definimos en base a sus atributos, distinguiendo entre **esquema de relación** (*cabecera*) y **relación** (*cuerpo*): un esquema de relación se compone de un nombre de relación R, de un conjunto de *n* atributos $\{A_i\}$ Y de un conjunto de *n* dominios (no necesariamente distintos) $\{D_i\}$, donde cada atributo será definido sobre un dominio:

$$R\ (A_1: D_1, A_2: D_2, \ldots A_n: D_n)$$

Una relación r(R) es un conjunto de m elementos denominados tuplas $\{t_j\}$. Cada $tupla_j$ es conjunto de pares $(<A_1:v_{1j}>,...<A_{ij}:v_{ij}>,...<A_n:v_{nj}>$ donde cada $\mathbf{A_i}$ es el nombre de un atributo y $\mathbf{v_{ij}}$ es un valor del correspondiente dominio $\mathbf{D_i}$ sobre el que está definido el atributo:

$$r(R) = \{t_j\ (<A_1:v_{1j}>,...<A_{ij}:v_{ij}>,...<A_n:v_{nj}>): V_{ij} \in D_i\}$$

Para establecer la cabecera (el esquema) de la relación EMPLEADO primero se definen los dominios, que tienen un nombre y un tipo de datos asociado:

- Dominio NUME_EMPLE, conjunto de 9 caracteres, de los cuales 8 son dígitos y el último es una letra.
- Dominio NOMBRES, conjunto de 25 caracteres.
- Dominio PAGA, conjunto de 4 dígitos.
- Dominio DEPART, posibles valores de números de departamento, rango de 01 a 99.
- Dominio FECHAS, conjunto de 10 caracteres.

La cabecera de la tabla EMPLEADO sería:

{ (NºEmple:NUME_EMPLE), (Apellidos:NOMBRES), (Salario: PAGA),
(Numdepart:DEPART), (FechaAlta:FECHAS) }

Una de las tuplas es:

{ (NºEmple: 13407877B), (Apellidos: Milagros Suela Sarro), (Salario: 1500),
(Numdepart:10) , (FechaAlta: 18/11/1990) }

Una relación se representa utilizando una tabla donde:

- Las columnas de la tabla son los atributos que expresan las propiedades de la relación. El número de atributos se llama **grado de la relación**.
- Cada fila de la tabla, llamada tupla, es un elemento del conjunto que es la relación. El número de tuplas se llama **cardinalidad de la relación**. La cardinalidad varía en el transcurso del tiempo.

No se deben confundir los conceptos de tabla y de relación, puesto que:

- Una tabla es una forma de representar una relación. Ambas deben tener:
  - Un nombre único que las distinga de las demás.
  - No pueden tener dos atributos iguales.

- Pero, una relación tiene unas propiedades intrínsecas que no tiene una tabla, y que se derivan de la misma definición matemática de relación, ya que, al tratarse de un conjunto, en una relación:
  - 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 único valor del dominio simple subyacente; no se admiten grupos repetitivos (ni otro tipo de estructuras) como valores de los atributos de una tupla: los atributos son atómicos.

3. INTENSIÓN Y EXTENSIÓN DE UNA RELACIÓN

De acuerdo con la definición de relación y esquema de relación dados anteriormente, se pueden distinguir dos conceptos ligados a la noción de relación:

- ***Intensión de una relación***: Parte definitoria y estática (invariante en el tiempo) de la relación, es lo que llamaremos *esquema de relación*.
- **Extensión**: Conjunto de tuplas que, en un instante determinado, satisfacen el esquema de relación y se encuentran almacenadas en la base de datos; es lo que se suele llamar, simplemente, relación. La extensión varía en el transcurso del tiempo.

En la Fig. 3 se puede ver un ejemplo de intensión y extensión de una relación.

INTENSIÓN DE UNA RELACIÓN:

CURSO (*Cód_curso*: ***Códigos***, *Nombre*: ***Nombres***, *N_Horas*: ***Horas***, *Materia*: ***Materias***)

EXTENSIÓN DE UNA RELACIÓN:

CURSO

Cód_curso	Nombre	*N_horas*	*Materia*
00012	DISEÑO DE BASES DE DATOS	50	Bases de Datos
00034	BASES DE DATOS ORIENTADAS OBJETOS	30	Bases de Datos
00167	SISTEMAS OPERATIVOS AVANZADOS	30	Sis. Operativos
01521	ALMACENES DE DATOS	25	Bases de Datos
005142	INTRODUCCIÓN AL C++	25	Lenguajes

Fig. 3. Ejemplo de intensión y extensión de una relación.

La intensión de una base de datos relacional, llamada **esquema relacional**, está compuesta por una colección de esquemas de relación que describen un determinado universo del discurso; la extensión del esquema relacional, constituido por una colección de relaciones, es la base de datos relacional.

4. ELEMENTOS NO PERMITIDOS: RESTRICCIONES

En todos los modelos de datos existen restricciones que a la hora de diseñar una base de datos se tienen que tener en cuenta. Los datos almacenados en la base de datos han de adaptarse a las estructuras impuestas por el modelo y deben cumplir una serie de reglas para garantizar que son correctos. El modelo relacional impone dos tipos de restricciones (algunas de las cuales ya las hemos citado):

- Restricciones inherentes al modelo
- Restricciones semánticas o de usuario.

4.1. Restricciones inherentes

Son las restricciones derivadas de la misma estructura del modelo, que no tienen que ser definidas por el usuario e imponen limitaciones a la hora de modelar nuestro mundo real.

La estructura del modelo relacional, al disponer solamente de un constructor, la relación, no permite diferenciar entre entidades e interrelaciones entre ellas.

Otras restricciones inherentes del modelo relacional provienen de su misma definición matemática, ya que, al definirse la relación como un conjunto, no se permite la existencia de tuplas duplicadas (un conjunto no puede tener elementos iguales) y de ahí la *obligatoriedad* de una clave primaria o identificador (conjunto mínimo de atributos que identifican de forma unívoca las tuplas de una relación). Es preciso advertir que esta restricción inherente al modelo no suele ser de los productos (SGBD), ya que, aunque en general permiten la definición de clave primaria, ésta no es obligatoria, es decir, se permiten tuplas duplicadas; lo mismo ocurre con el estándar SQL 92.

También la definición matemática de relación impone restricciones inherentes como la prohibición de que en el cruce de una fila y una columna haya más de un valor, es decir, las relaciones son tablas planas (de dos dimensiones) no admitiéndose los grupos repetitivos (en una celda cualquiera no puede haber dos valores, deben ser atómicos); ya hemos dicho que cada atributo sólo puede tomar un valor del dominio simple subyacente (es lo que se conoce como primera forma normal, como veremos al estudiar la teoría de la normalización).

Además, el orden de las tuplas y de los atributos no es significativo (el orden de los elementos de un conjunto es irrelevante), por lo que operadores basados en un orden (que existen en otros modelos de datos) no tienen razón de ser en el modelo relacional.

Integridad de Entidad

Es una restricción inherente, debida asimismo a la necesidad de que todas las tuplas de una relación sean distintas, la cual establece que: ***"Ningún atributo que forme parte de la clave primaria puede tomar valores nulos"***. El valor nulo (NULL) de un atributo representa información desconocida, inaplicable, etc. Por lo tanto, si alguno de los atributos que forman parte de la clave primaria tomase valores nulos, algunas de las tuplas de la relación no podrían ser identificadas, por lo que se violaría la condición de que el valor de la clave debe ser único para cada tupla.

4.2. Restricciones semánticas

Las restricciones semánticas son facilidades proporcionadas por el modelo a fin de poder recoger en el esquema de relación la semántica del universo del discurso que estamos modelando. Son restricciones que tiene que definir el diseñador a fin de que el esquema sea un reflejo lo más fiel posible del mundo real; de ahí la importancia de las restricciones en el diseño de BD. Es preciso advertir que las restricciones semánticas que ofrece el modelo relacional no siempre coinciden con las del SQL ni con las de los productos. Nosotros, para describirlas, nos apoyaremos en el SQL, añadiendo los disparadores.

En el modelo relacional, de forma análoga a lo que ocurre en general en un modelo de datos, una restricción de integridad es una regla ECA (Evento-Condición-Acción), donde el evento es una operación de actualización (inserción, borrado o modificación), la condición puede definirse como un predicado sobre un conjunto de atributos, de tuplas o de dominios, que debe ser verificado por los correspondientes elementos para que constituyan una extensión válida del esquema, y la acción puede ser de rechazo de la operación o cualquier otra, determinada por el modelo o bien por el usuario; la acción se lleva a cabo si una operación de actualización intenta violar la condición.

La implementación de las restricciones de integridad en un sistema relacional exige facilidades de definición de restricciones, así como procedimientos que impidan que en la base de datos aparezcan estados inconsistentes, es decir, procedimientos que pongan en marcha las correspondientes acciones a fin de mantener la consistencia.

Por tanto, un Sistema de Gestión de Bases de Datos Relacional (SGBDR), al igual que otros sistemas no relacionales, debe incluir facilidades que permitan:

En la fase de definición:

- Describir las restricciones con precisión y sencillez.
- Indicar las acciones ante una posible violación de una restricción.
- Verificar la consistencia de las restricciones entre sí mismas.

En fase de manipulación:

- Comprobar que las actualizaciones cumplen las restricciones de integridad.
- Poner en marcha las acciones indicadas en el caso de que las restricciones no se cumplan.

Las restricciones semánticas hacen que las ocurrencias de los esquemas de la base de datos sean válidas. Los mecanismos que proporciona el modelo para este tipo de restricciones son los siguientes:

a) La **restricción de clave primaria** (*PRIMARY KEY*) permite declarar uno o varios atributos como clave primaria de una relación. Esta restricción suele implicar las dos siguientes: de unicidad y obligatoriedad.

b) La **restricción de unicidad** (*UNIQUE*) permite definir claves alternativas. Los valores de los atributos no pueden repetirse.

c) La **restricción de obligatoriedad** (*NOT NULL*) permite declarar si uno o varios atributos no pueden tomar valores nulos.

d) **Integridad referencial o restricción de clave ajena** (*FOREIGN KEY*). Se utiliza para enlazar relaciones, mediante claves ajenas, de una base de datos. La integridad referencial indica que los valores de la clave ajena en la relación hijo se corresponden con los de la clave primaria en la relación padre.

Una **clave ajena** *de una relación R2 es un conjunto no vacío de atributos cuyos valores han de coincidir con los valores de la clave primaria de una relación R1 (R1 y R2 pueden ser la misma relación) o ser nulos*. Se dice que R2 es la relación *que referencia (hijo),* mientras que R1 es la relación *referenciada (padre).*

En la Fig. siguiente se muestra un ejemplo de clave ajena, donde los valores del atributo *Cód-prog* de la relación CURSO_DOCTORADO (relación que referencia) deben coincidir con los de la clave primaria de la relación PROGRAMA (relación referenciada).

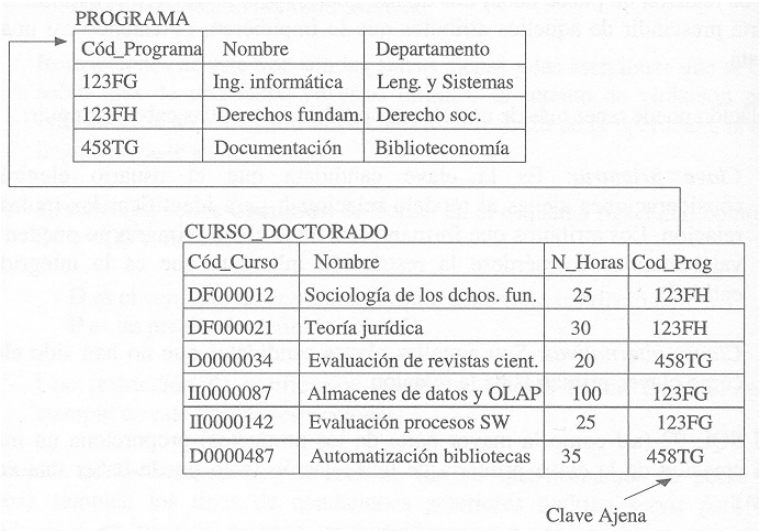


Fig. 4. Ejemplo de Integridad Referencial mediante una clave ajena.

En la Fig. 5 se muestra otro ejemplo de clave ajena en la que los valores del atributo COD_JEFE de la relación EMPLEADOS deben existir entre los valores ya introducidos en el atributo COD_EMP (un jefe es un empleado), y cada uno de los valores contenidos en el atributo COD_DEP de la misma relación deben coincidir con alguno de los valores existentes en ese momento en el COD_DEP de la relación DEPARTAMENTOS.

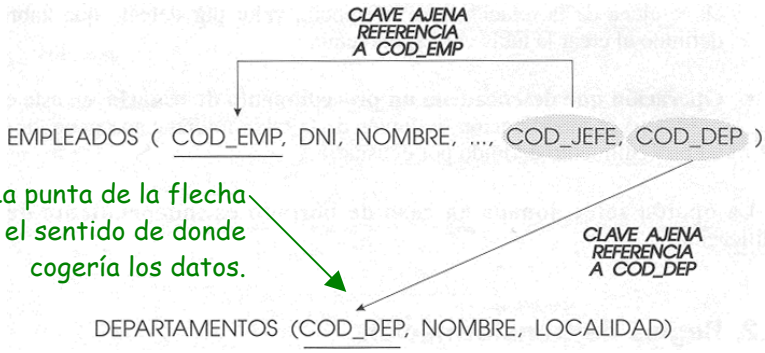


Fig. 5. Ejemplo de clave ajena.

Además de definir las claves ajenas hay que tener en cuenta las operaciones de borrado y/o actualización que se realizan sobre las tuplas de la relación referenciada. Las posibilidades son las siguientes:

- **Borrado y/o modificación en cascada** (*CASCADE*). El borrado o modificación de una tupla en la relación padre (relación con la clave primaria) ocasiona un borrado o modificación de las tuplas relacionadas en la relación hija (relación que contiene la clave ajena). En el caso de empleados y departamentos, si se borra un departamento de la tabla TDEPART se borrarán los empleados que pertenecen a ese departamento. Igualmente ocurrirá si se modifica el NUMDEPT de la tabla TDEPART esa modificación se arrastra a los empleados que pertenezcan a ese departamento.
- **Borrado y/o modificación restringido** (*RESTRICT, NO ACTION*). En este caso no es posible realizar el borrado o la modificación de las tuplas de la relación padre si existen tuplas relacionadas en la relación hija. Es decir, no podría borrar un departamento que tiene empleados.
- **Borrado y/o modificación con puesta a nulos** (*SET NULL*). Esta restricción permite poner la clave ajena en la tabla referenciada a NULL si se produce el borrado o modificación en la tabla primaria o padre. Así pues, si se borra un departamento, a los empleados de ese departamento se les asignará NULL en el atributo NUMDEPT.
- **Borrado y/o modificación con puesta a valor por defecto** (*SET DEFAULT*). En este caso, el valor que se pone en las claves ajenas de la tabla referenciada es un valor por defecto que se habrá especificado en la creación de la tabla.

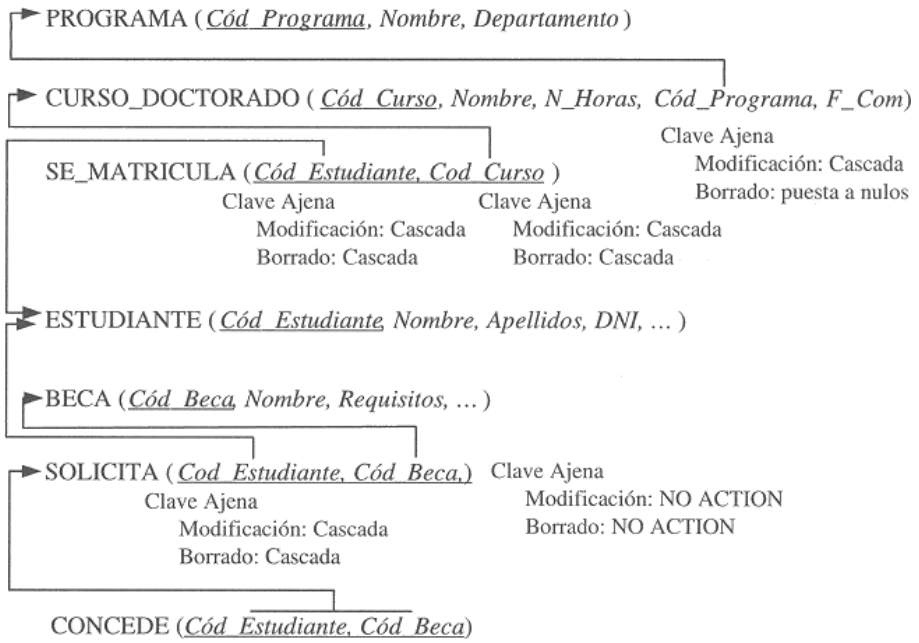


Fig. 6. Ejemplos de definición de claves ajenas con sus opciones

e) **Restricciones de verificación** (*CHECK*). Esta restricción permite especificar condiciones que deban cumplir los valores de los atributos. Cada vez que se realice una inserción o una actualización de datos se comprueba si los valores cumplen la condición. Rechaza la operación si no se cumple.

Ejemplo: Si en el esquema de la Fig. 6 se quisiera imponer la restricción de que los cursos de doctorado tuvieran un mínimo de 30 horas, se definiría una restricción de verificación en la tabla CURSO_DOCTORADO:

```
CHECK N_Horas ≥ 30
```

A continuación, se muestran dos sentencias de creación de tablas, la tabla FABRICANTES (tabla relacionada) y la tabla ARTÍCULOS (tabla que relaciona). Un fabricante fabrica muchos artículos. Observa las restricciones: (en PL-SQL de Oracle)

```
CREATE TABLE FABRICANTES (
  CD_FAB NUMBER (3) NOT NULL DEFAULT 100,
  NOMBRE VARCHAR2(15) UNIQUE,
  PAIS VARCHAR2(15) CONSTRAINT CK_PA CHECK (PAIS=UPPER(PAIS)) ,
  CONSTRAINT PK_FA PRIMARY KEY (CD_FAB),
  CONSTRAINT CK_NO CHECK (NOMBRE=UPPER(NOMBRE)) );
```

```
CREATE TABLE ARTICULOS (
  ARTIC VARCHAR2(20) NOT NULL,
  COD_FA NUMBER(3) NOT NULL,
  PESO NUMBER(3) NOT NULL CONSTRAINT CK1_AR
     CHECK (PESO>0),
  CATEGORIA VARCHAR2(10) NOT NULL,
  PRECIO_VENTA NUMBER (4) CONSTRAINT CK2_AR
     CHECK(PRECIO_VENTA>O) ,
  PRECIO_COSTO NUMBER (4) CONSTRAINT CK3_AR
     CHECK(PRECIO_COSTO>0),
  EXISTENCIAS NUMBER (5),
  CONSTRAINT PK_ART PRIMARY KEY (ARTIC, COD_FA, PESO, CATEGORIA),
  CONSTRAINT FK_ARFA FOREIGN KEY (COD_FA) REFERENCES
     FABRICANTES (CD_FAB)
        ON DELETE CASCADE,
  CONSTRAINT CK_CAT
     CHECK (CATEGORIA IN ('Primera', 'Segunda', 'Tercera'))) ;
```

f) **Aserciones** (*ASSERTION*). Son parecidas a la anterior, pero en este caso en lugar de afectar a una sola relación como CHECK, puede afectar a dos o más relaciones. En este caso la condición se establece sobre elementos de distintas relaciones. Pueden implicar a subconsultas en la condición. La definición de una aserción debe tener un nombre. Tiene vida por sí misma.

Ejemplo: Supongamos, siguiendo el mismo ejemplo, que en la relación CONCEDE entre BECA y ESTUDIANTE se desea indicar que solamente se pueda conceder una beca a uno de los estudiantes que ha solicitado dicha beca; para conseguirlo, se podría definir la siguiente aserción en el esquema relacional:

```
CREATE ASSERTION CONCEDE_SOLICITA AS
   CHECK (SELECT Cod_Estudiante, Cod_Beca
      FROM CONCEDE) IN
         (SELECT Cod_Estudiante, Cod_Beca
            FROM SOLICITA));
```

Esta aserción sustituiría la clave ajena que habríamos definido, ya que haría exactamente lo mismo que hace la clave ajena: impedir que cualquier tipo de actualización deje en la relación CONCEDE tuplas que no estaban en SOLICITA

g) **Disparadores** (*TRIGGER*). Las restricciones anteriores son declarativas, sin embargo, este tipo es procedimental. El usuario podrá especificar una serie de acciones distintas ante una determinada condición. El usuario escribe el procedimiento a aplicar dependiendo del resultado de la condición. Los disparadores están soportados a partir de los estándares SQL3.

A continuación, se muestra un disparador de base de datos que audita las operaciones de inserción y borrado de datos en la tabla EMPLE. Cada vez que se realiza una operación de actualización o borrado se inserta en la tabla AUDITAREMPLE una fila que contendrá: la fecha y hora de la operación, el número y apellido del empleado afectado, y la operación que se realiza.

```
CREATE OR REPLACE TRIGGER auditar_act_emp
   BEFORE INSERT OR DELETE ON EMPLE FOR EACH ROW
BEGIN
   IF DELETING THEN
      INSERT INTO AUDITAREMPLE
         VALUES (TO_CHAR (sysdate, 'DD/MM/YY*HH24:MI*')
            || : OLD.EMP_NO || '*' || : OLD.APELLIDO || '* BORRADO' );
   ELSIF INSERTING THEN
      INSERT INTO AUDITAREMPLE
         VALUES(TO_CHAR (sysdate, 'DD/MM/YY*HH24:MI*')
            || :NEW.EMP_NO || '*' || :NEW.APELLIDO || '* INSERCION ');
   END IF;
END;
```

Este tipo de restricciones es muy importante en el diseño de bases de datos a fin de no perder en el esquema relacional aquella semántica del esquema conceptual que no es posible recoger mediante las anteriores restricciones.

Para ilustrar con otro ejemplo el uso de disparadores, vamos a suponer que si una beca es solicitada por más de 50 alumnos, se introduce un texto en una tabla de mensajes para que, la persona que gestiona las becas, considere si es necesario ofrecer más becas.

```
CREATE TRIGGER Comprobar_Matriculados
   AFTER INSERT ON SOLICITA
DECLARE
   NUM_SOLICITUDES Number;
BEGIN
   SELECT COUNT(*) INTO NUM_SOLICITUDES FROM SOLICITA;
   IF NUM_SOLICITUDES > 50 THEN
      INSERT INTO MENSAJES VALUES ('Hay más de 50 solicitudes');
   END IF;
END Comprobar_Matriculados;
```

h) También se pueden utilizar los dominios para realizar restricciones: **restricciones sobre los valores de los dominios**. En este caso estamos limitando los posibles valores de un atributo limitando los valores posibles en el dominio sobre el que está definido.

Por ejemplo: Podemos limitar los valores del atributo ALTURA a sólo tres valores si definimos el dominio ALTURAS como "alto", "mediano" y "bajo".

5. LOS TRES NIVELES DE ANSI EN EL MODELO RELACIONAL

El modelo relacional responde a la parte lógica de la arquitectura a tres niveles de ANSI. El esquema conceptual de ANSI será el esquema relacional y los esquemas externos se corresponden con las vistas. El modelo relacional es un modelo lógico y, por tanto, no contempla los aspectos físicos del nivel interno de ANSI, que son propios de los productos; por ello, el SQL92 no tiene ninguna sentencia de tipo físico como, por ejemplo, "CREATE INDEX".

5.1. El nivel conceptual del modelo relacional: Esquema de Relación y Esquema Relacional

El *esquema de relación* como intensión (definición) de una relación, es una descripción de la misma, donde, además de los atributos (con referencia a los correspondientes dominios) sobre los que está definida la relación, es preciso especificar también las restricciones de integridad que deben cumplir las tuplas de la relación para ser ejemplares válidos de dicho esquema.

Por tanto, un esquema de relación se define como:

R (A:D, S)

Donde:

- **R** es el nombre de la relación.
- **A** es el conjunto de atributos, cada uno de los cuales está definido sobre un dominio **D**.
- **S** son las restricciones de integridad intrarrelación (sobre atributos o sobre tuplas).

Una extensión válida de un esquema de relación es una r*elación base*, definida sobre el conjunto de atributos de dicho esquema, donde cada atributo toma sus valores del correspondiente dominio y que satisface todas sus restricciones de integridad.

El *esquema relacional*, como intensión de una base de datos relacional, es una descripción de la misma donde, además del conjunto de esquemas de relación, es preciso especificar los dominios y las restricciones de integridad interrelación y sobre dominios, además de las vistas.

Por tanto, un esquema relacional se define como:

E (R, D, T, V)

Conjunto de relaciones

Donde:

- **E** es el nombre del esquema relacional
- **R** es el conjunto de esquemas de relación.
- **D** es la definición del conjunto de dominios.
- **T** es el conjunto de restricciones de integridad interrelación y sobre dominios.
- **V** es el conjunto de vistas.

Una extensión del esquema relacional es el conjunto de valores de los dominios que forman parte del esquema relacional, más el conjunto de extensiones de los esquemas de relación que lo componen y que satisfacen todas sus restricciones (conjunto de relaciones base).

5.2. Las vistas y el nivel externo en el modelo relacional

El nivel externo en el modelo relacional está constituido, además de por las relaciones base (tablas), por las *vistas*, las cuales son relaciones derivadas que se definen dando un nombre a una expresión de consulta. Se podría decir que las vistas son *relaciones virtuales* (como "ventanas" sobre otras relaciones), en el sentido de que no tienen datos almacenados, sino que lo único que se almacena es su definición en términos de otras relaciones.

Las vistas son importantes en el diseño de bases de datos porque proporcionan a los usuarios una forma de ver los datos más sencilla y apropiada a las necesidades de sus aplicaciones. Las vistas facilitan además la definición de restricciones de confidencialidad.

Una diferencia importante entre la arquitectura ANSI y el modelo relacional es que en éste la visión del usuario no está limitada a una vista, sino que un esquema externo de ANSI en el modelo relacional puede estar formado por un conjunto de vistas y/o de tablas base, es decir, los usuarios pueden "ver" directamente las tablas base, además (o en lugar) de las vistas.

Las vistas se definen mediante una sentencia del lenguaje de definición de datos ("***CREATE VIEW***") que contiene una expresión de consulta, la cual no se ejecuta en el momento de su definición, sino posteriormente cuando se invoca la vista al aparecer su nombre en una consulta (en una sentencia ***SELECT***) o en una actualización (inserción→***INSERT***, borrado →***DELETE***, o modificación ***UPDATE***).

5.3. El nivel interno en el modelo relacional

El nivel interno de una base de datos describe cómo se encuentran los datos almacenados en el soporte físico. Es un objetivo prioritario del diseño físico minimizar el número de accesos al soporte donde se encuentran los datos almacenados físicamente.

El modelo relacional es un modelo lógico y, por tanto, como ya hemos señalado anteriormente, sólo contempla los dos niveles lógicos: el conceptual (relaciones base) y el externo (relaciones base y vistas), por lo que no se puede hablar propiamente del nivel interno en el modelo relacional. Sin embargo, sí es preciso que los SGBDR tengan los datos físicamente almacenados en memoria secundaria (no sólo las tablas sino también las estructuras complementarias, como índices, punteros, direcciones de páginas, etc., a fin de conseguir un acceso más eficiente); la descripción de todos estos datos almacenados constituye el esquema interno de la arquitectura ANSI.

5.4. Correspondencia de la arquitectura ANSI y el modelo relacional.

En la Fig. 7 se puede observar, como resumen de lo que acabamos de exponer, la correspondencia entre los tres niveles de la arquitectura ANSI y el modelo relacional.

ANSI		RELACIONAL	
LOGICO	Nivel Externo	SQL	Vistas Relaciones Base
	Nivel Conceptual		Relaciones Base
FISICO	Nivel Interno	PRODUCTOS	Datos Almacenados - Relaciones base almacenadas - Índices - Punteros - Direcciones de página - ...

Fig. 7. Resumen de la correspondencia entre niveles.

6. TRANSFORMACIÓN DE UN ESQUEMA ENTIDAD/RELACION A UN ESQUEMA RELACIONAL

6.1. Introducción

Las tres reglas básicas para convertir un esquema en el modelo E/R al relacional son las siguientes:

1) Todo tipo de entidad se convierte en una relación.
2) Para todo tipo de interrelación 1:N se realiza lo que se denomina propagación de clave (regla general), o bien se crea una nueva relación.
3) Todo tipo de interrelación N:M se transforma en una relación.

Debido a que el modelo relacional no distingue entre entidades e interrelaciones, ambos conceptos deben representarse mediante relaciones. Esto implica una pérdida de semántica con respecto al esquema E/R, ya que las interrelaciones N:M no se distinguen de las entidades y las 1:N se representan mediante una propagación de clave, desapareciendo incluso el nombre de la interrelación.

En el ejemplo de la Fig. 8 puede observarse que las tres entidades, DEPARTAMENTO, PROFESOR y CURSO se han transformado en otras tantas relaciones. La interrelación N:M *Imparte* da lugar a una nueva relación cuya clave es la concatenación de las claves primarias de las entidades que participan en ella (Cód_prof de PROFESOR y Cód_curso de CURSO), siendo además éstas claves ajenas de *Imparte*, que referencian a las tablas PROFESOR y CURSO, respectivamente; la interrelación 1:N *Pertenece* se ha transformado mediante el mecanismo de propagación de clave, por el que se ha incluido en la tabla PROFESOR el atributo clave de la entidad DEPARTAMENTO (Nombre_dep), que constituye, por tanto, clave ajena de la relación PROFESOR referenciando a la tabla DEPARTAMENTO.

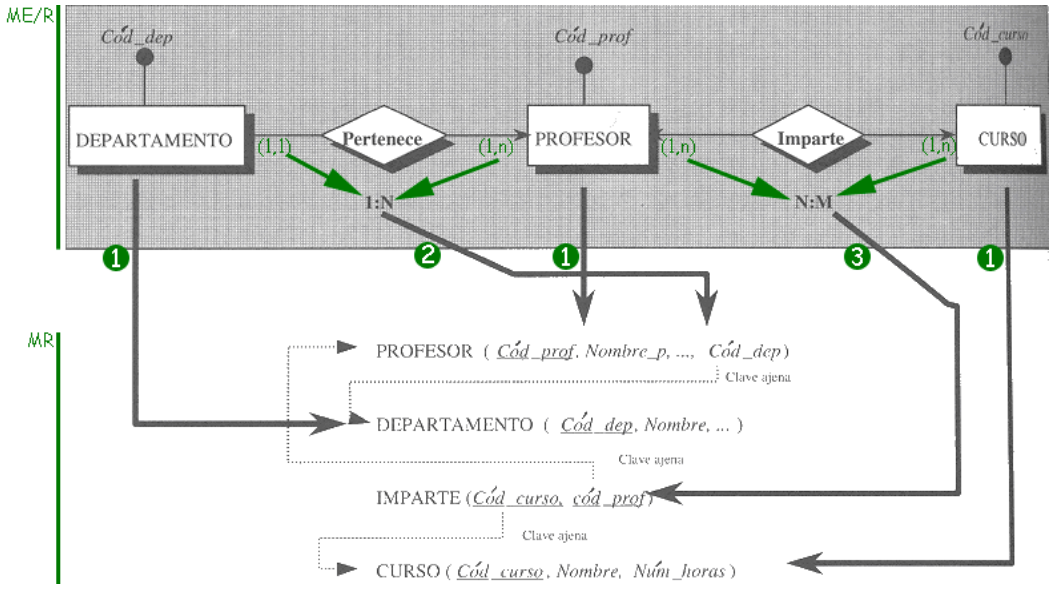


Fig. 8. Ejemplo de paso del Modelo Entidad Relación al modelo relacional

Teniendo en cuenta que el modelo E/R básico tiene otros objetos y que, además, hemos propuesto una serie de extensiones, hemos de ver cómo se puede recoger el modelo E/R completo (con sus extensiones) en el modelo relacional y señalar, en su caso, aquellas características que, debido a la menor semántica del modelo relacional, no es posible representar directamente, a fin de implementarlas posteriormente por medio de disparadores, procedimientos almacenados o procedimientos de usuario, con los problemas que en este último caso pueden aparecer relativos al mantenimiento de la integridad del sistema.

6.2. Reglas concernientes al Modelo Básico

1. **Transformación de dominios**. En el modelo relacional estándar un dominio es un objeto más, propio de la estructura del modelo que, como tal, tendrá su definición concreta en el Lenguaje de Definición de Datos LDD (en nuestro caso el SQL92) que se elija. Como ejemplo podemos crear el dominio de los estados civiles, que es un conjunto de valores de tipo carácter, de longitud 1, que puede tomar los valores 'S', 'C', 'V' o 'D' (véase Fig. 9).

   En el SQL propuesto expresaríamos este dominio de la siguiente forma:

```
CREATE DOMAIN
  Estados_Civiles AS CHAR(1)
  CHECK (VALUE IN ('S', 'C', 'V', 'D'))
```

ME/R

MR
DOMINIO G_CIVIL CHAR(1)
CHECK (VALUE IN ('S', 'C', 'V', 'D')

Fig. 9. Transformación de dominios

2. **Transformación de entidades**. Según lo que hemos indicado en la introducción de este capítulo, "*cada tipo de entidad se convierte en una relación*". Esto es, el modelo lógico estándar posee el objeto RELACION o TABLA mediante el cual representamos las entidades. La tabla se llamará igual que el tipo de entidad de donde proviene. Para su definición disponemos en el SQL de la sentencia *CREATE TABLE*. Por ejemplo, la entidad PROFESOR se transforma en una tabla con ese mismo nombre (véase Fig. 10). En este caso la transformación es directa, y no hay pérdida de semántica.

3. **Transformación de atributos de entidades**. "*Cada atributo de una entidad se transforma en una columna de la relación a la que ha dado lugar la entidad*". Pero teniendo en cuenta que tenemos atributos identificador principal, otros que son identificadores alternativos y el resto de atributos que no son identificadores (atributos no principales) desglosamos esta regla en tres subreglas:

3.1. Atributos identificadores. El (o los) atributo(s) que son identificador(es) principales (AIP) "*pasan a ser la clave primaria de la relación*". Por ejemplo, en la Fig. 10 tenemos la relación PROFESOR, fruto de la trasformación de la entidad del mismo nombre, con su identificador principal (AIP) *Cod_prof* que pasa a ser la clave primaria.

El Lenguaje Lógico Estándar (LLS) recoge directamente este concepto por medio de la cláusula **PRIMARY KEY** en la descripción de la tabla, luego la transformación es directa y no hay pérdida de semántica.

ME/R

MR

PROFESOR

Cód_prof	*Nombre*	*DNI*	*Dirección*	*Teléfono*	*Materia*
00001	Juan	12223433	Ríos Rosas, 23	670123123	Ing. Software
00002	Coral	54656754	Alarcos, 8	567983456	Bases de datos
00003	Belén	53567523	Getafe, 4	6°9267854	Orientación objetos
00004	Goyo	97856757	Pez, 102	679345763	Sistemas operativos
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
03568	Roberto	34534522	Fundación, 10	639456239	Redes

CLAVE PRIMARIA

PROFESOR (*Cód_prof, Nombre, DNI, dirección, Teléfono, Materia*)

Fig. 10. Transformación de una entidad

3.2. Atributos identificadores alternativos. Respecto a los atributos identificadores alternativos el Lenguaje Lógico Estandar (LLS) recoge por medio de la cláusula ***UNIQUE*** estos objetos. Al ser la transformación directa, no hay pérdida de semántica. Si se desea que estos atributos no tomen valores nulos habrá que indicarlo.

"*Los identificadores alternativos pasan a estar identificados como únicos*".

3.3. **Atributos no identificadores**. Estos atributos "*pasan a ser columnas de la relación*", como los anteriores, las cuales tienen permitido tomar valores nulos a no ser que se indique lo contrario.

Aplicando las reglas 2 y 3, la transformación de la entidad PROFESOR al modelo relacional estándar se representa mediante el Lenguaje Lógico Estandar (LLS).

```
CREATE TABLE Profesor (
    Cód_Profesor Códigos,
    Nombre Nombres,
    DNI DNIS NOT NULL
    Dirección Lugares,
    Teléfono Nos_Teléfono,
    Materia Materias,
    PRIMARY KEY (Cód_Profesor),
    UNIQUE (DNI));
```

4. **Transformación de interrelaciones**. Ya hemos señalado que, dependiendo del tipo de correspondencia de la interrelación, y de otros aspectos semánticos de la misma variará la manera de realizar la transformación al esquema relacional, por eso desglosamos esta regla en tres subreglas:

   4.1. **Interrelaciones N:M**. Un tipo de interrelación N:M "*se transforma en una relación que tendrá como clave primaria la concatenación de los identificadores principales (AIP) de los tipos de entidad que asocia*". Se ve claramente, como ya hemos indicado anteriormente, que dentro de un esquema relacional no hay manera de diferenciar qué relaciones provienen de una entidad y cuáles de ellas proceden de la transformación de interrelaciones, por lo tanto hay cierta pérdida de semántica en este punto de la transformación; semántica que sólo puede ser salvada, almacenando comentarios sobre la procedencia de cada una de las tablas o mediante un convenio en la denominación de estas tablas que provienen de interrelaciones en el modelo Entidad Relación.

   Por ejemplo, la Fig. 11 muestra esta transformación, en la que se presenta la asociación que existe entre los profesores y los cursos que imparten, apareciendo una relación cuya clave primaria está compuesta por la concatenación del código del profesor y el código del curso.

   Además, cada uno de los atributos que forman la clave primaria de esta relación son claves ajenas que referencian a las tablas en que se han convertido las entidades interrelacionadas (claves primarias), lo que se especifica en el Lenguaje Lógico Estándar (LLS) a través de la cláusula ***FOREIGN KEY*** dentro de la sentencia de creación de la tabla. Habrá que estudiar, además, qué ocurre en los casos de borrado y modificación de la clave primaria referenciada, teniendo en cuenta que en nuestro Lenguaje Lógico Estándar (LLS) las opciones permitidas son operación restringida (en caso de no especificar la acción o poner ***NO ACTION***), puesta a nulo (***SET NULL***), puesta a valor por defecto (***SET DEFAULT***) u operación en cascada (***CASCADE***).

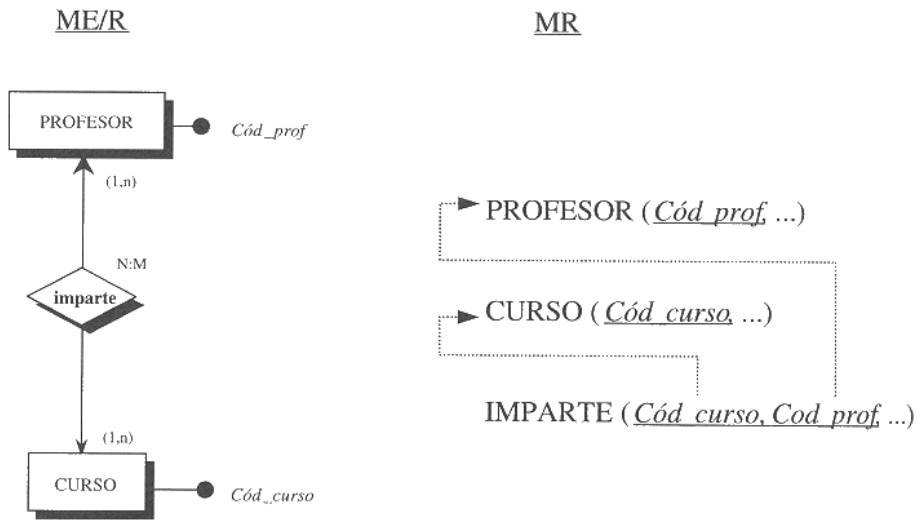


Fig. 11. Transformación de una interrelación N:M

Aplicando lo que acabamos de decir al ejemplo de la Fig. 11, obtendríamos en SQL:

```
CREATE TABLE Imparte (
    Cód_Profesor Codigos_P,
    Cód_Curso Codigos_C,
    …,
    PRIMARY KEY (Cód_Profesor, Cód_Curso),
    FOREIGN KEY (Cód_Profesor) REFERENCES Profesor
       ON DELETE CASCADE
       ON UPDATE CASCADE,
    FOREIGN KEY (Cód_Curso) REFERENCES Curso
       ON DELETE CASCADE
       ON UPDATE CASCADE)
```

Sería también correcto no dar las opciones de borrado y modificación (o lo que es lo mismo, poner ***NO ACTION***), en cuyo caso se rechazarían el borrado o modificación de aquellas tuplas de las tablas referenciadas cuando su valor de la clave primaria existiese en la tabla que referencia. En cambio, no se admitirían las opciones de puesta a nulos o a valor por defecto.

Otra característica que debemos recoger en esta transformación es la cardinalidad mínima de cada una de las entidades que participan en la interrelación, lo que se hace mediante la especificación de restricciones, aserciones o disparadores.

Un ejemplo de este caso sería aquel en el que suponemos que cada curso puede ser impartido por menos de 4 profesores, que podría quedar reflejado en la siguiente aserción:

```
CREATE ASSERTION Profesor_Curso
CHECK NOT EXIST (SELECT COUNT(*)
    FROM IMPARTE
    GROUP BY COD_CURSO
    HAVING COUNT(*) ≥ 4)
```

4.2. **Interrelaciones 1:N**. Existen dos soluciones para la transformación de una interrelación 1:N:

a) Propagar los Identificador(es) Principales (AIP) del tipo de entidad que tiene de cardinalidad máxima 1 a la que tiene N, es decir en el sentido de la flecha, desapareciendo el nombre de la interrelación, con lo cual se pierde semántica (ésta es la regla habitual). Es decir, "*Se propagan los Identificadores Principales (AIP) y los atributos a la relación que tiene la n*". Podemos ver un ejemplo en la Fig. 12.

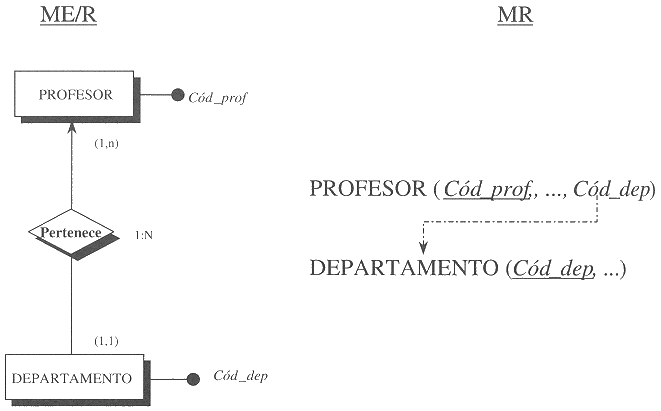


Fig. 12. Transformación de una interretación 1:N en propagación de clave

b) Transformarla en una relación, como si se tratara de una interrelación N:M; sin embargo en este caso, la clave primaria de la relación creada es sólo la clave primaria de la tabla a la que le corresponde la cardinalidad *N*.

Aunque depende del criterio del diseñador, e influye además de la semántica, la eficiencia, los casos en los que puede ser apropiado transformar la interrelación en una relación son los siguientes:

1) Cuando el número de ejemplares interrelacionados de la entidad que propaga su clave es muy pequeño y, por tanto, existirían muchos valores nulos en la clave propagada. Por ejemplo, en la Fig. 13, en principio, existirían dos soluciones, pero si suponemos que el número de subtemas que pertenecen a un tema es muy pequeño en comparación con los que son independientes, puede no ser conveniente propagar la clave de tema a los que dependen de él, ya que aparecerían muchos valores nulos; por tanto, la solución b) puede ser adecuada (por el solo hecho de existir algún ejemplar no interrelacionado la cardinalidad mínima es cero).

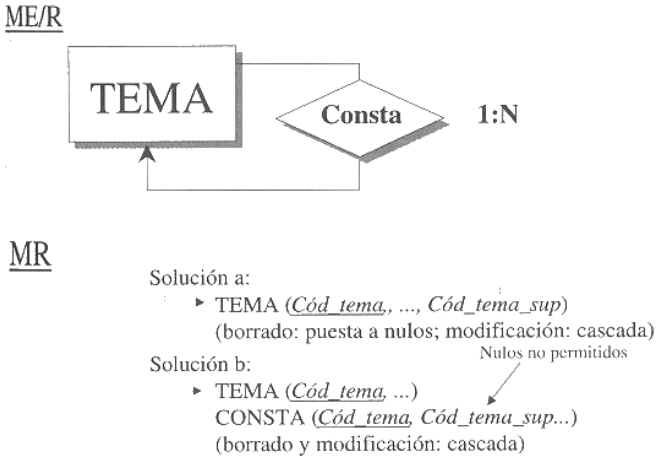


Fig. 13. Transformación de una interrelación 1:N en una relación

2) Cuando se prevé que dicha interrelación en un futuro se convertirá en una de tipo N:M.
3) Cuando la interrelación tiene atributos propios y no deseamos propagarlos (a fin de conservar la semántica).

Por otro lado, la propagación de clave causa la aparición de claves ajenas, con sus mecanismos de borrado y actualización correspondientes, según la semántica del problema.

Si transformamos la interrelación de la Fig. 13 en una relación (solución b), las correspondientes sentencias en SQL serían:

```
CREATE TABLE Consta (
    Cód_Tema Codigos,
    Cód_Tema Sup Codigos,
    PRIMARY KEY (Cód_tema)
    FOREIGN KEY (Cód_Tema) REFERENCES Tema
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREING KEY (Cod_Tema_sup) REFERENCES Tema
        ON DELETE CASCADE
        ON UPDATE CASCADE);
```

Si se transforma la interrelación en una relación, el control de las reglas de borrado y modificación se realiza de forma análoga a la de la regla 4.1. Si se utiliza el mecanismo de propagación de clave, la cardinalidad mínima de la entidad para la cual la cardinalidad máxima es uno (entidad de nivel superior) se puede controlar impidiendo o permitiendo la existencia de nulos en la clave ajena propagada.

Como se puede observar en la Fig. 14, al ser la cardinalidad de DEPARTAMENTO (1,1), no pueden admitirse valores nulos en la clave propagada; sin embargo sí habrían de admitirse si la cardinalidad fuera (0,1).

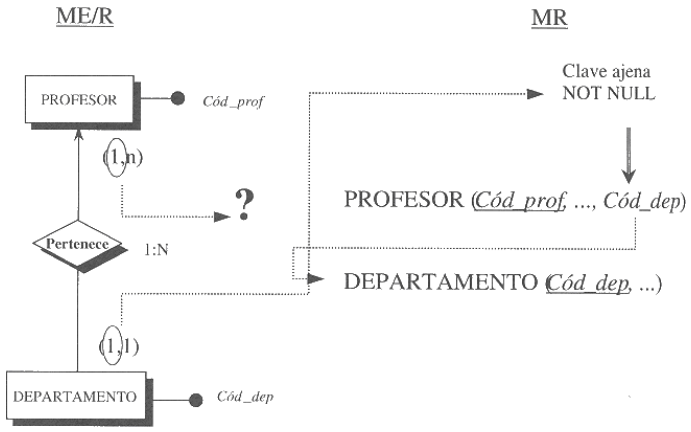


Fig. 14. Transformación de cardinalidades mínimas

La cláusula ***NOT NULL*** no resuelve el problema de la cardinalidad mínima 1 en la entidad en la que se incluye la clave propagada debiéndose definir para controlarla la correspondiente restricción (***check***, ***aserción*** o ***disparador***).

4.3. **Interrelaciones 1:1**. Una interrelación de tipo 1:1 es un caso particular de una N:M o, también, de una 1:N, por lo que no hay regla fija para la transformación de este tipo de interrelación al modelo relacional estándar, pudiéndose aplicar la regla 4.1 (con lo que crearíamos una relación), es decir, "*creamos una nueva relación y propagamos los Identidicadores Principales IAP, poniendo uno como clave y los demás como únicos y no nulos*" o aplicar la regla 4.2. (esto es, propagar la clave correspondiente). En este último caso hay que observar que en una interrelación 1:1, la propagación de la clave puede efectuarse en ambos sentidos.

Los criterios para aplicar una u otra regla y para propagar la clave se basan en las cardinalidades mínimas, en recoger la mayor cantidad de semántica posible, evitar los valores nulos o en motivos de eficiencia. A continuación exponemos algunos ejemplos que puedan servir de orientación:

a) Si las entidades que se asocian poseen cardinalidades (0,1), puede ser conveniente transformar la interrelación 1:1 en una relación. Por ejemplo, en la Fig. 15 tenemos la interrelación *Matrimonio* entre tipos de entidad HOMBRE y MUJER, la cual se transformará en una relación, evitando así los valores nulos que aparecerían en caso de propagar la clave de MUJER a la tabla HOMBRE o viceversa, ya que como reflejan las cardinalidades no todos los hombres ni todas las mujeres se encuentran casados.

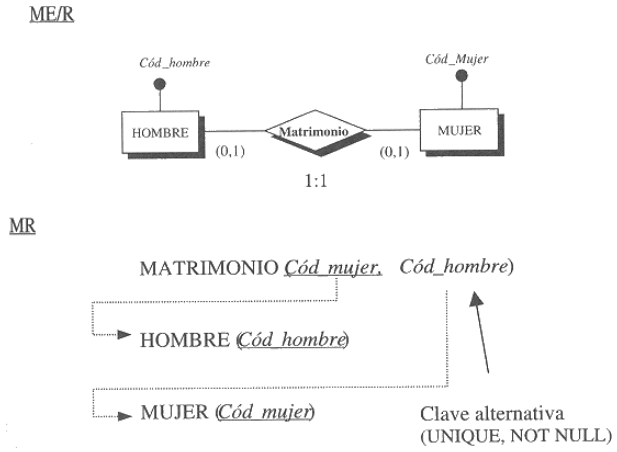


Fig. 15. Transformación de una interrelación 1:1 en una relación

b) Si una de las entidades que participa en la interrelación posee cardinalidades (0,1), mientras que en la otra son (1,1), conviene propagar la clave de la entidad con cardinalidades (1,1) a la tabla resultante de la entidad de cardinalidades (0,1). En la Fig. 16 tenemos una interrelación que recoge que un profesor es responsable de un departamento, y supone, que un profesor puede ser responsable como máximo de un departamento y como mínimo de ninguno y que cada departamento tiene que tener siempre un responsable (pero sólo uno), en este caso propagamos la clave de PROFESOR a la tabla de DEPARTAMENTO, evitando así valores nulos y captando más semántica (recogemos la cardinalidad mínima 1, que en caso de realizar la propagación en sentido contrario no podríamos captar directamente).

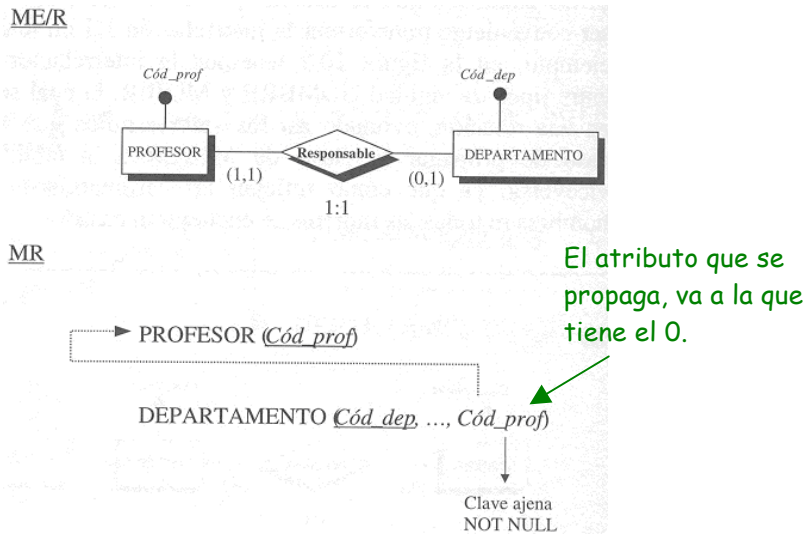


Fig. 16. Transformación de una interrelación 1:1 por propagación de clave

c) En el caso de que ambas entidades presenten cardinalidades (1,1), se puede propagar la clave de cualquiera de ellas a la tabla resultante de la otra, teniendo en cuenta en este caso los accesos más frecuentes y prioritarios a los datos de las tablas. Se puede plantear (también por motivos de eficiencia) la propagación de las dos claves, lo que introduce redundancias que deben ser controladas por medio de restricciones.

5. **Transformación de atributos de interrelaciones**. "*Si la interrelación se transforma en una relación, todos sus atributos pasan a ser columnas de la relación*". Por ejemplo, la interrelación Imparte entre PROFESOR y CURSO de la Fig. 10 tiene un atributo Núm_horas (número de horas que imparte) que pasa a ser una columna de la tabla que se crea a partir de ella. La transformación es directa y no hay pérdida de semántica.

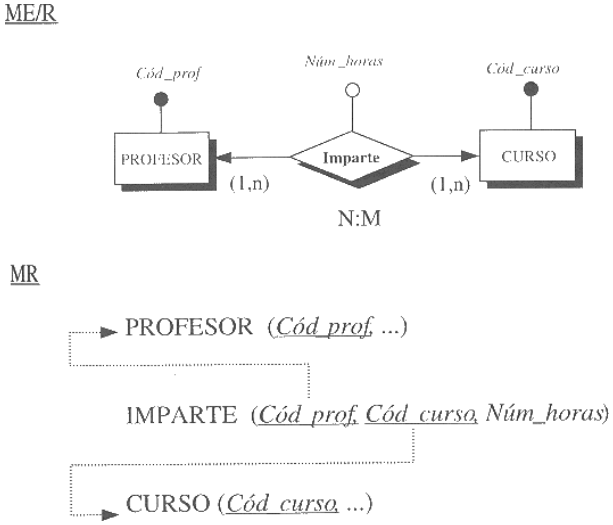


Fig. 17. Transformación de los atributos de una interrelación en columnas de una tabla

En caso de que la interrelación se transforme mediante propagación de clave, sus atributos migran junto a la clave a la relación que corresponda, aunque ya hemos advertido que en este caso puede ser preferible crear una nueva relación para representar las interrelaciones que tienen atributos.

6. **Transformación de restricciones**. En cuanto a "*las restricciones de usuario, existen ciertas cláusulas en el Lenguaje Lógico Estándar (LLS) que pueden recogerlas*". Por ejemplo, podemos restringir a un rango determinado los valores de un dominio a través de la cláusula ***BETWEEN***, o bien determinar por enumeración los valores que puede tomar una columna en una tabla con la cláusula ***IN***, como podemos observar en la regla 1 (véase Fig. 9).

   Otra posibilidad es utilizar la cláusula ***CHECK*** dentro de la descripción de la tabla para expresar una condición que deben cumplir un conjunto de atributos de la tabla, o la cláusula aserción si la comprobación afecta a atributos de más de una tabla. Por ejemplo, para que la fecha de inicio de un curso sea siempre menor que la de finalización, en la creación de la tabla tendríamos las siguientes sentencias SQL:

```
CREATE TABLE Curso (
    Cód_Curso Cursos,
    Nombre Nombres,
    Num_Horas Horas
    Fecha_I Fechas,
    Fecha_F Fechas,
    PRIMARY KEY (Cód_Curso),
    CHECK (Fecha_I < Fecha_F));
```

   Existe, además, la posibilidad de utilizar disparadores que, si bien no existen en el SQL92, sí se ofrecen en algunos productos.

6.3. Reglas concernientes a las extensiones del Modelo Entidad Relación

7. **Transformación de dependencias en identificación y en existencia**. Las dependencias en existencia y en identificación no son recogidas directamente en el Modelo Lógico Estándar (MLS). En el ejemplo de la Fig. 18 vemos que la manera de transformar una interrelación de este tipo es utilizar el mecanismo de propagación de clave, creando una clave ajena, con nulos no permitidos, en la relación de la entidad dependiente, con la característica de obligar a una modificación y un borrado en cascada.

   Además, en el caso de dependencia en identificación la clave primaria de la relación en la que se ha transformado la entidad débil debe estar formada por la concatenación de las claves de las dos entidades participantes en la interrelación.

   La clave ajena de la nueva relación, si es en existencia no necesita ser clave, pero si es en identificación necesita ser clave principal de la nueva relación.

   Así, el esquema E/R de la Fig. 18 da lugar al esquema relacional en SQL:

```
CREATE TABLE Curso (
  Cód_Curso Códigos_cursos,
  …,
  PRIMARY KEY (Cód_Curso));
```

```
CREATE TABLE Edición (
  Cód_Curso Códigos_Cursos,
  Cód_Edición Códigos_Ediciones,
  …,
  PRIMARY KEY (Cód_Curso, Cód_Edicion)
  FOREIGN KEY (Cód_Curso) REFERENCES Curso
     ON DELETE CASCADE
     ON UPDATE CASCADE);
```

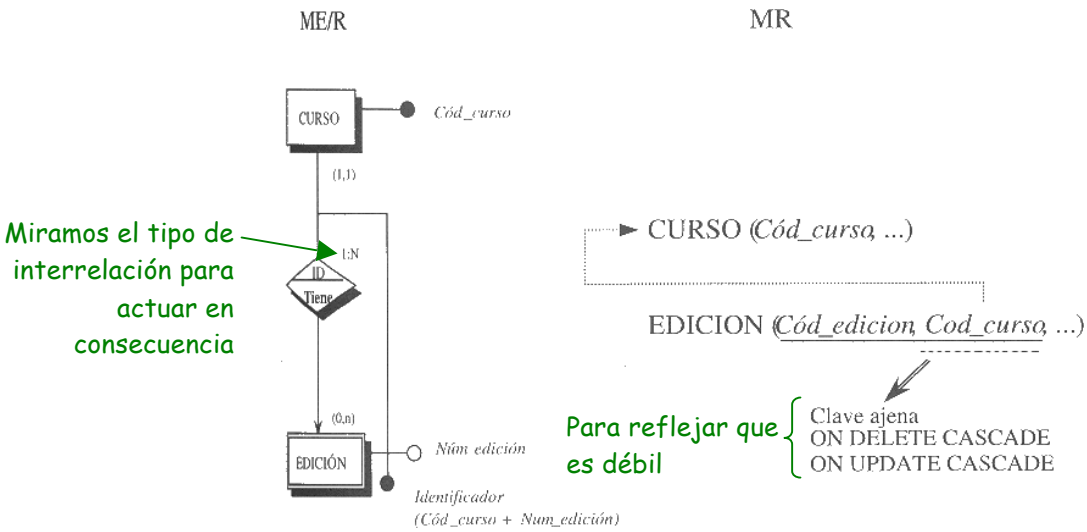


Fig. 18. Transformación de una dependencia en identificación

8. **Transformación de restricciones de interrelaciones.** Para soportar restricciones de interrelaciones (exclusión, inclusión, etc.) debemos definir las restricciones pertinentes en cada caso. Por ejemplo, en la Fig. 19 se muestra una restricción de exclusividad donde un profesor puede dirigir o impartir cursos, pero no ambas cosas. Las interrelaciones *Dirige* e *Imparte* las resolvemos mediante el mecanismo de propagación de clave, llevando *cód_prof_dirige* y *cód_prof_imparte* a la relación CURSO.

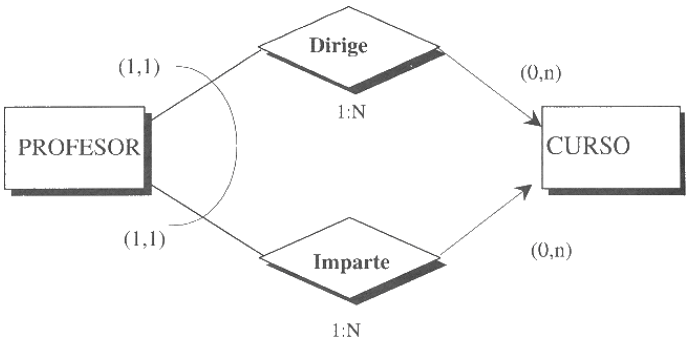


Fig. 19. Restricción de exclusividad de interrelaciones

Para hacer que se cumpla la exclusividad habría que introducir las correspondientes restricciones en una cláusula *CHECK*, tal como se indica a continuación:

```
CREATE TABLE Curso (
  Cód_Curso Códigos_Cursos,
  Nombre Nombres,
  …,
  Cód_prof_dirige Códs_profesores,
  Cód_prof_imparte Códs_profesores,
  PRIMARY KEY (Cód_Curso)
  FOREIGN KEY (Cód_prof_dirige) REFERENCES Profesor
     ON UPDATE CASCADE
  FOREIGN KEY (Cód_prof_imparte) REFERENCES Profesor
     ON UPDATE CASCADE
  CHECK ((Cód_prof_dirige NOT IN (SELECT Cód_prof_imparte FROM CURSO)
     AND Cod_prof_imparte NOT IN (SELECT Cód_prof_dirige FROM CURSO))
);
```

Dejamos como ejercicio al lector resolver otros casos de interrelaciones exclusivas 1:N y de interrelaciones exclusivas N:M, así como otros tipos de restricciones entre interrelaciones, mediante la definición de CHECK y/o de aserciones de forma análoga al ejemplo anterior.

9. **Transformación de supertipos y subtipos (de las generalizaciones).** En lo que respecta a los supertipos y subtipos, no son objetos que se puedan representar explícitamente en el modelo relacional. Ante un tipo de entidad y sus subtipos caben varias soluciones de transformación al modelo relacional, con la consiguiente pérdida de semántica dependiendo de la estrategia elegida. Destacamos tres:

   a) Englobar todos los atributos de la entidad y sus subtipos en una sola relación. En general, adoptaremos esta solución cuando los subtipos se diferencien en muy pocos atributos y las interrelaciones que los asocian con el resto de las entidades del esquema sean las mismas para todos (o casi todos) los subtipos. Por ejemplo, la diferencia que existe entre un profesor que sea doctor y otro que no lo sea, podemos considerarla mínima teniendo en cuenta que ambos tienen los mismos atributos y ambos podrán impartir cursos (ver Fig. 20). Por este motivo, la solución adecuada en este caso sería la creación de una sola tabla que contenga todos los atributos del supertipo y los de los subtipos, añadiendo el atributo discriminante que indica el tipo de profesor.

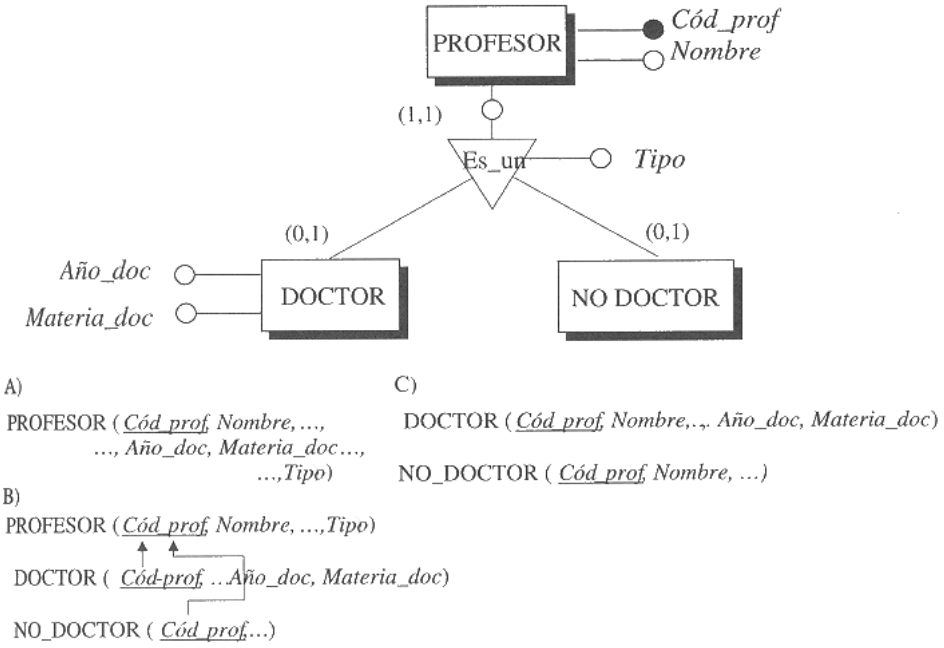


Fig. 20. Transformación de subtipos

También habrá que especificar las restricciones semánticas correspondientes, por ejemplo:

```
CHECK ((Tipo = 'NO_DOCTOR'
  AND Año_Doc IS NULL
  AND Materia_Doc IS NULL))
  OR (Tipo= "DOCTOR"
  AND Año_doc IS NOT NULL
  AND Materia IS NOT NULL));
```

Hay que observar que el atributo discriminante de la jerarquía podrá admitir valores nulos en el caso de que la jerarquía sea parcial y que deberá declararse como *NOT NULL* si la jerarquía es total. Por otra parte, el atributo discriminante constituirá un grupo repetitivo, si los subtipos se solapan, debiendo, por tanto, separar este atributo en una relación aparte que tendrá como clave la concatenación de la clave del supertipo con el atributo discriminante; otra solución bastante más eficiente consiste en crear un código para los valores del atributo discriminante que contemple los posibles subtipos solapados.

b) Crear una relación para el supertipo y tantas relaciones como subtipos haya, con sus atributos correspondientes. Ésta es la solución adecuada cuando existen muchos atributos distintos entre los subtipos y se quieren mantener de todas maneras los atributos comunes a todos ellos en una relación. Al igual que en el caso anterior, habrá que crear las restricciones y/o aserciones oportunas.

c) Considerar relaciones distintas para cada subtipo, que contengan, además de los atributos propios, los atributos comunes. Se elegiría esta opción cuando se dieran las mismas condiciones que en el caso anterior (muchos atributos distintos) y los accesos realizados sobre los datos de los distintos subtipos siempre afectan a atributos comunes.

Podemos, por tanto, elegir entre tres estrategias distintas para la transformación de un supertipo y sus subtipos al modelo relacional. Sin embargo, desde un punto de vista exclusivamente semántico, la opción b es la mejor. Por otra parte, desde el punto de vista de la eficiencia tenemos que tener en cuenta que:

a) El acceso a una fila que refleje toda la información de una determinada entidad es mucho más rápido (no hace falta combinar varias relaciones).

b) Es la menos eficiente, aunque, como ya hemos señalado, es la mejor desde un punto de vista exclusivamente semántico.

c) Con esta solución aumentamos la eficiencia ante determinadas consultas (las que afecten a todos los atributos, tanto comunes como propios, de un subtipo) pero la podemos disminuir ante otras. Esta solución es en la que se pierde más semántica; además si existe solapamiento se introduce redundancia que debe ser controlada si queremos evitar inconsistencias. Hay que tener en cuenta que esta solución no es válida cuando la generalización es parcial.

Elegiremos una estrategia u otra dependiendo de que sea la semántica o la eficiencia la que prime para el usuario en un momento determinado.

Por lo que se refiere a la totalidad/parcialidad de la jerarquía y al solapamiento/disyunción de los subtipos, pueden ser soportados por medio de *CHECK* o *aserciones*, así como por disparadores o por procedimientos almacenados.

10. **Transformación de la dimensión temporal.** En el caso de que en el esquema Entidad Relación aparezca el tiempo como un tipo de entidad, la transformación en el esquema relacional estándar no constituye mayor problema, ya que se tratará como otro tipo de entidad cualquiera y, por tanto se creará una relación más:

TIEMPO (FechaI, Fecha_F, HoraI, Hora_F, MinutosI, ...)

Sin embargo, cuando la dimensión temporal la hemos recogido en el esquema Entidad Relación a través de atributos de interrelación de tipo FECHA, la transformación en el Modelo Lógico Estándar (MLS) consiste en pasarlos a columnas de la relación que corresponda. Sobre este punto debemos tener cuidado a la hora de elegir la clave primaria de la relación resultante, dependiendo de los supuestos semánticos del entorno.

Por ejemplo, en la Fig. 21 podemos comprobar que la clave primaria de la relación obtenida de la interrelación "un socio toma prestado un libro de la biblioteca durante un periodo de tiempo determinado" no es sólo la concatenación del IDENTIFICADOR(ES) PRINCIPALES (AIP) de SOCIO y el de LIBRO, sino que si se supone que un socio puede tomar prestado un mismo libro en distintos periodos de tiempo, es necesario, a fin de formar la clave primaria, añadir a los códigos de SOCIO y de LIBRO el atributo F _inicio. La transformación, por lo tanto, es directa, pero debemos tener en cuenta lo que acabamos de señalar respecto a la clave primaria a fin de salvaguardar la integridad de la base de datos.

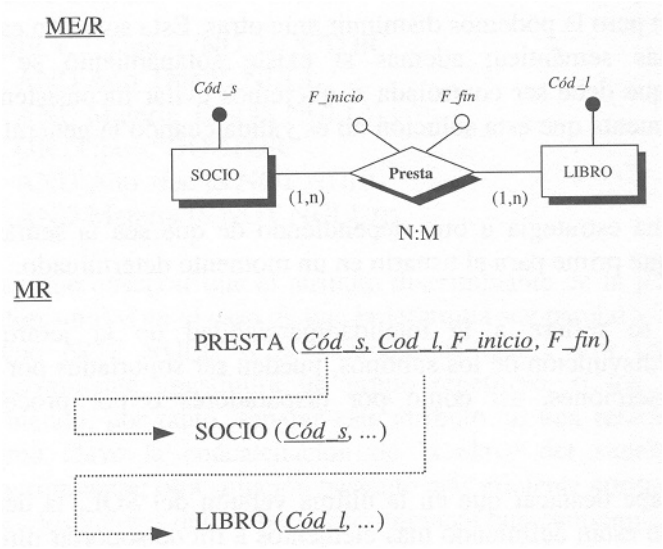


Fig. 21. Transformación de la dimensión temporal

Es preciso observar que esta aparente sencillez en la transformación de algo tan complicado como es la dimensión temporal es debida a la poca semántica y precisión del modelo E/R (al igual que en el modelo relacional) para tratar los aspectos temporales. En sistemas de información en los que la dimensión temporal es fundamental, como es el caso de los sistemas estadísticos, el modelo E/R no resuelve muchos de los problemas de modelado relativos a la dimensión temporal.

11. **Transformación de Atributos Derivados.** No existe para los atributos derivados una representación directa y concreta en el Modelo Lógico Estandard (MLS), sino que se pueden tratar como atributos normales, que pasarán a ser columnas de la relación que corresponda (véase Fig. 22).

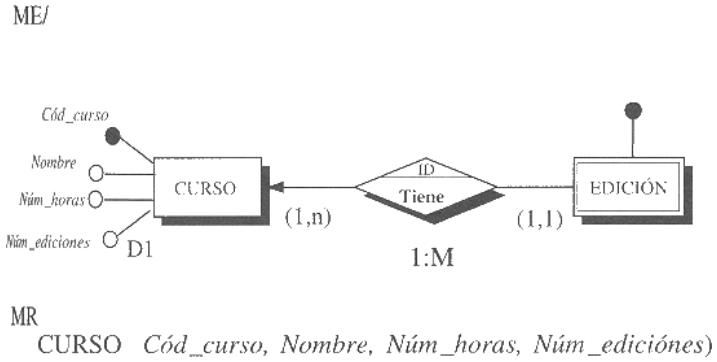


Fig. 22. Transformación de atributo derivado

En este caso es preciso construir un disparador que calcule el valor del atributo derivado cada vez que se inserten o borren las ocurrencias de los atributos que intervienen en el cálculo de éste y añadir las restricciones correspondientes. Por ejemplo, en el caso de la Fig. 15, cada vez que se inserte o borre una tupla en la tabla EDICIÓN, el disparador debe actualizar el atributo *N_ediciones* de la tabla CURSO.

Otra solución es no almacenar las columnas que provengan de atributos derivados, creando procedimientos que calculen los valores de éstos cada vez que se recuperan, lo que en ocasiones puede plantear problemas de semántica y en ocasiones de eficiencia.

6.4. Grafo relacional

Una forma de representar gráficamente el esquema relacional de una manera sencilla y completa es el denominado *grafo relacional, diagrama esquemático o grafo de combinación*.

Es un grafo compuesto de un conjunto de nodos, donde cada nodo representa un esquema de relación, es decir, una tabla de la base de datos.

Para cada tabla, como mínimo, ha de aparecer su nombre y sus atributos, indicando su clave primaria (subrayando los atributos que la componen con trazo continuo) y sus claves ajenas (subrayando los correspondientes atributos por trazo discontinuo), véase Fig. 23.

Se dibuja, además, un conjunto de arcos que conectan los atributos que constituyen la clave ajena con la tabla referenciada, permitiendo así que el usuario entienda los campos clave que comparten dominios comunes; en definitiva, los arcos representan la *referenciabilidad* de los atributos (clave ajena) de una relación respecto a la clave primaria de la otra. Nosotros proponemos que los arcos estén direccionados de modo que el arco parta de la clave ajena y la flecha señale a la tabla referenciada.

Se debería indicar también qué debe hacer el SGBD en el caso de modificar o borrar una tupla en la tabla principal (referenciada) con las tuplas de la tabla "subordinada" (la que tiene la clave ajena) que estén relacionadas con la tupla modificada o borrada.

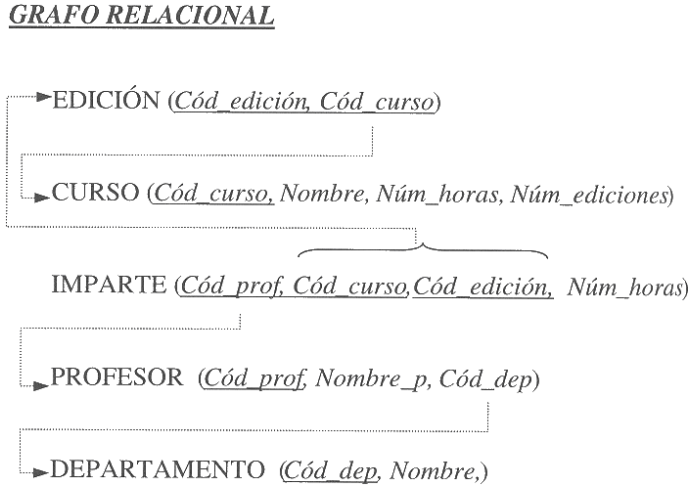


Fig. 23. Ejemplo de grafo relacional