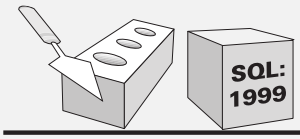


3

EL MODELO RELACIONAL

- ¿Cómo se representan los datos en el modelo relacional?
- ¿Qué restricciones de integridad se pueden expresar?
- ¿Cómo se pueden crear y modificar datos?
- ¿Cómo se pueden manipular y consultar los datos?
- ¿Cómo se pueden crear, modificar y consultar tablas empleando SQL?
- ¿Cómo se obtiene el diseño de una base de datos relacional a partir de su diagrama ER?
- ¿Qué son las vistas y por qué se emplean?
- **Conceptos fundamentales:** relación, esquema, ejemplar, tupla, campo, dominio, grado, cardinalidad; LDD de SQL, `CREATE TABLE, INSERT, DELETE, UPDATE`; restricciones de integridad, restricciones de dominio, restricciones de clave, `PRIMARY KEY, UNIQUE`, restricción de clave externa, `FOREIGN KEY`; mantenimiento de la integridad referencial, restricciones diferidas e inmediatas; consultas relacionales; diseño lógico de bases de datos, traducción de los diagramas ER a relaciones, expresión de las restricciones ER mediante SQL; vistas, vistas e independencia lógica, seguridad; creación de vistas en SQL, actualización de vistas, consultas sobre vistas, eliminación de vistas.

TABLA: Disposición de palabras, números, signos o combinaciones de éstos en columnas paralelas para mostrar una serie de hechos o de relaciones de forma clara, compacta y completa; sinopsis o esquema.

— *Diccionario Webster de la lengua inglesa*

Codd propuso el modelo relacional de datos en 1970. En ese momento la mayor parte de los sistemas de bases de datos se basaban en dos modelos de datos más antiguos (el modelo jerárquico y el de red); el modelo relacional revolucionó el campo de las bases de datos y

SQL. Desarrollado originalmente como lenguaje de consulta del SGBD relacional pionero System-R de IBM, el lenguaje estructurado de consultas (structured query language, SQL) se ha convertido en el lenguaje más utilizado para la creación, manipulación y consulta de SGBD relacionales. Dado que muchos fabricantes ofrecen productos SQL, existe la necesidad de una norma que defina el "SQL oficial". La existencia de una norma permite que los usuarios evalúen la completitud de la versión de SQL de cada fabricante. También permite que los usuarios distingan las características de SQL propias de un producto de las que están normalizadas; las aplicaciones que se basan en características no normalizadas son menos portables.

La primera norma de SQL la desarrolló en 1986 el Instituto Nacional Americano de Normalización (American National Standards Institute, ANSI), y se denominó SQL-86. Hubo una pequeña revisión en 1989 denominada SQL-89 y una más importante en 1992 denominada SQL-92. La Organización Internacional para la Normalización (International Standards Organization, ISO) colaboró con ANSI en el desarrollo de SQL-92. Actualmente la mayor parte de los SGBD comerciales soportan (el subconjunto principal de) SQL-92, y se trabaja para que soporten la versión de la norma SQL:1999 recientemente adoptada, una importante ampliación de SQL-92. El tratamiento de SQL en este libro se basa en SQL:1999, pero es aplicable también a SQL-92; las características exclusivas de SQL:1999 se señalan de manera explícita.

sustituyó en gran parte a los modelos anteriores. A mediados de los años setenta del siglo veinte se desarrollaron prototipos de sistemas relacionales de administración de bases de datos en proyectos de investigación pioneros de IBM y de UC-Berkeley y varios fabricantes ya ofrecían productos de bases de datos relacionales poco después. Hoy en día, el modelo relacional es el modelo de datos dominante y la base de los productos SGBD líderes, incluidos la familia DB2 de IBM, Informix, Oracle, Sybase, Access y SQLServer de Microsoft, FoxBase y Paradox. Los sistemas relacionales de bases de datos son ubicuos en el mercado y representan una industria de muchos miles de millones de euros.

El modelo relacional es muy sencillo y elegante: cada base de datos es un conjunto de *relaciones*, cada una de las cuales es una tabla con filas y columnas. Esta representación tabular tan sencilla hace que incluso los usuarios más novatos puedan comprender el contenido de las bases de datos y permite el empleo de lenguajes sencillos de alto nivel para consultar los datos. Las principales ventajas del modelo relacional frente a los modelos de datos más antiguos son su sencilla representación de los datos y la facilidad con la que se pueden formular incluso las consultas más complejas.

Aunque este libro se centre en los conceptos subyacentes, también se presentarán las características del **lenguaje de definición de datos (LDD)** de SQL, el lenguaje estándar para la creación, manipulación y consulta de los datos en los SGBD relacionales. Esto permitirá anclar firmemente la discusión en términos de los sistemas reales de bases de datos.

El concepto de relación se discute en el Apartado 3.1 y se muestra la manera de crear relaciones empleando el lenguaje SQL. Un componente importante de los modelos de datos es el conjunto de estructuras que ofrecen para la especificación de las condiciones que deben

cumplir los datos. Esas condiciones, denominadas *restricciones de integridad* (RI), permiten que los SGBD rechacen las operaciones que puedan corromper los datos. Las restricciones de integridad del modelo relacional se presentan en el Apartado 3.2, junto con una discusión del soporte de las RI en SQL. La manera en que los SGBD hacen que se cumplan las restricciones de integridad se discute en el Apartado 3.3.

En el Apartado 3.4 se examina el mecanismo para tener acceso a los datos y recuperarlos de la base de datos (los *lenguajes de consulta*) y se presentan las características que proporciona SQL para la consulta, que se estudiarán con más detalle en un capítulo posterior.

A continuación se discute la conversión de los diagramas ER en esquemas de las bases de datos relacionales en el Apartado 3.5. Se presentan las *vistas*, o tablas definidas mediante consultas, en el Apartado 3.6. Las vistas se pueden emplear para definir el esquema externo de una base de datos y así ofrecer el soporte para la independencia lógica de los datos en el modelo relacional. En el Apartado 3.7 se describen las órdenes SQL para la eliminación y modificación de tablas y vistas.

Finalmente, en el Apartado 3.8 se amplía el estudio del caso de diseño, la tienda en Internet presentada en el Apartado 2.8, mostrando el modo en que el diagrama ER de su esquema conceptual se puede traducir al modelo relacional, y también la manera en que el empleo de vistas puede ayudar en este diseño.

3.1 INTRODUCCIÓN AL MODELO RELACIONAL

La principal estructura para la representación de datos en el modelo relacional son las **relaciones**. Cada relación consiste en un **esquema de relación** y un **ejemplar de relación**. El ejemplar de la relación es una tabla, y el esquema de la relación describe las cabeceras de las columnas de esa tabla. En primer lugar se describirá el esquema de la relación y, posteriormente, el ejemplar de la relación. El esquema especifica el nombre de la relación, el de cada **campo** (o **columna**, o **atributo**), y el **dominio** de cada campo. En el esquema de relación se hace referencia al dominio por su **nombre de dominio** y tiene un conjunto de **valores** asociados.

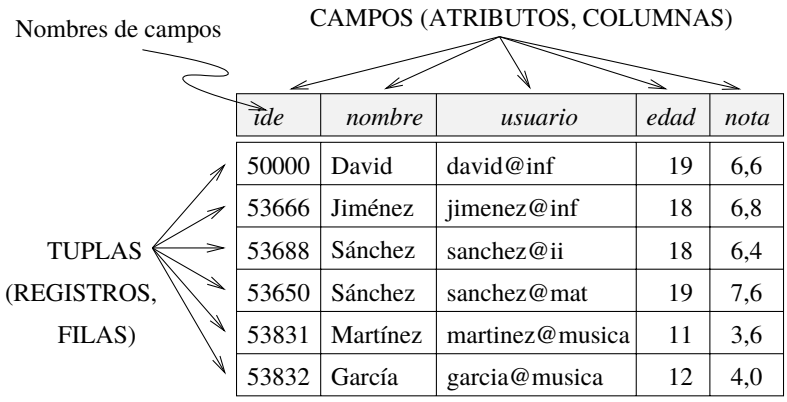
Para ilustrar las partes del esquema de una relación se empleará el ejemplo introducido en el Capítulo 1 sobre la información de alumnos en la base de datos de una universidad:

Alumnos(*ide:* `string`, *nombre:* `string`, *usuario:* `string`,
edad: `integer`, *nota:* `real`)

Esto indica, por ejemplo, que el campo denominado *ide* tiene un dominio denominado `string`. El conjunto de valores asociado con el dominio `string` es el conjunto de todas las cadenas de caracteres.

Examinemos ahora los ejemplares de las relaciones. Cada **ejemplar** de una relación es un conjunto de **tuplas**, también denominadas **registros**, en el que cada tupla tiene el mismo número de campos que el esquema de la relación. Se puede pensar en cada ejemplar de una relación como en una *tabla* en la que cada tupla sea una *fila*, y todas las filas tienen el mismo número de campos. (El término *ejemplar de una relación* se suele abreviar a sólo *relación*, cuando no hay confusión posible con otros aspectos de la relación, como puede ser el esquema.)

En la Figura 3.1 se muestra un ejemplar de la relación Alumnos.



ide	*nombre*	*usuario*	*edad*	*nota*
50000	David	david@inf	19	6,6
53666	Jiménez	jimenez@inf	18	6,8
53688	Sánchez	sanchez@ii	18	6,4
53650	Sánchez	sanchez@mat	19	7,6
53831	Martínez	martinez@musica	11	3,6
53832	García	garcia@musica	12	4,0

Figura 3.1 El ejemplar $A1$ de la relación Alumnos

El ejemplar $A1$ contiene seis tuplas y tiene, como se esperaba del esquema, cinco campos. Obsérvese que no hay dos filas idénticas. Éste es un requisito del modelo relacional —cada relación se define como un *conjunto* de tuplas o filas únicas—.

En la práctica, los sistemas comerciales permiten que las tablas tengan tablas duplicadas, pero supondremos que cada relación es realmente un conjunto de tuplas a menos que se indique lo contrario. El orden en que aparecen las filas no es importante. La Figura 3.2 muestra el mismo ejemplar de la relación. Si los campos tienen nombre, como en las definiciones del

ide	*nombre*	*usuario*	*edad*	*nota*
53831	Martínez	martinez@musica	11	3,6
53832	García	garcia@musica	12	4,0
53688	Sánchez	sanchez@ii	18	6,4
53650	Sánchez	sanchez@mat	19	7,6
53666	Jiménez	jimenez@inf	18	6,8
50000	Díaz	diaz@inf	19	6,6

Figura 3.2 Una representación alternativa del ejemplar $A1$ de Alumnos

esquema y en las figuras que reflejan los ejemplares de la relación del ejemplo utilizado, tampoco importa el orden de los campos. No obstante, un convenio alternativo es relacionar los campos en un orden concreto y referirse a ellos por su posición. Así, *ide* es el campo 1 de Alumnos, *usuario* es el campo 3, etcétera. Si se emplea este convenio, el orden de los campos sí es significativo. La mayor parte de los sistemas de bases de datos emplea una combinación de estos convenios. Por ejemplo, en SQL se emplea el convenio de los campos con nombre en las instrucciones que recuperan tuplas, y la de los campos ordenados suele utilizarse al insertar tuplas.

El esquema de cada relación especifica el dominio de cada campo o columna del ejemplar de esa relación. Estas **restricciones de dominio** del esquema especifican una condición importante que se desea que satisfagan todos los ejemplares de la relación: los valores que aparecen en cada columna deben obtenerse del dominio asociado con esa columna. Así, el

dominio de cada campo es, esencialmente, el *tipo* de ese campo, en términos de los lenguajes de programación, y restringe los valores que pueden aparecer en él.

De manera más formal, sea R(c_1:D1, ..., c_n:Dn) el esquema de una relación y, para cada c_i, $1 \leq i \leq n$, sea Dom_i el conjunto de valores asociados con el dominio denominado Di. Cada ejemplar de R que cumple las restricciones de dominio del esquema es un conjunto de tuplas con n campos:

$$\{ \langle c_1 : d_1, \ \ldots \ , c_n : d_n \rangle \ \mid \ d_1 \in Dom_1, \ \ldots \ , d_n \in Dom_n \ \}$$

Los corchetes angulares $\langle \ldots \rangle$ identifican los campos de cada tupla. Empleando esta notación, la primera tupla de Alumnos mostrada en la Figura 3.1 se escribe $\langle$*ide:* 50000, *nombre:* Díaz, *usuario:* diaz@inf, *edad:* 19, *nota:* 6,6$\rangle$. Las llaves $\{\ldots\}$ denotan un conjunto (de tuplas, en esta definición). La barra vertical | significa "tal que", el símbolo $\in$ significa "pertenece a" y la expresión a la derecha de la barra vertical es una condición que deben cumplir los valores de los campos de cada tupla del conjunto. Por tanto, cada ejemplar de R se define como un conjunto de tuplas. Los campos de cada tupla deben corresponderse con los campos del esquema de la relación.

Las restricciones de dominio son tan fundamentales en el modelo relacional que a partir de aquí sólo se considerarán ejemplares que las satisfagan; por tanto, *ejemplar de la relación* significa *ejemplar de la relación que cumple las restricciones de dominio del esquema de la relación.*

El **grado**, también denominado **aridad**, de una relación es su número de campos. La **cardinalidad** de un ejemplar de la relación es el número de tuplas que contiene. En la Figura 3.1, el grado de la relación (el número de columnas) es cinco, y la cardinalidad de ese ejemplar es seis.

Una **base de datos relacional** es un conjunto de relaciones con diferentes nombres de relación. El **esquema de una base de datos relacional** es el conjunto de esquemas de las relaciones de la base de datos. Por ejemplo, en el Capítulo 1 se trató la base de datos de una universidad con las relaciones denominadas Alumnos, Profesores, Asignaturas, Aulas, Matriculado, Imparte e Impartida_en. Un **ejemplar** de una base de datos relacional es un conjunto de ejemplares de relaciones, uno por cada esquema de relación del esquema de la base de datos; evidentemente, cada ejemplar de relación debe cumplir las restricciones de dominio de su esquema.

3.1.1 Creación y modificación de relaciones mediante SQL

La norma del lenguaje SQL emplea la palabra *tabla* para denotar una *relación*, y ese convenio se seguirá a menudo en este libro al tratar de SQL. El subconjunto de SQL que soporta la creación, eliminación y modificación de tablas se denomina lenguaje de definición de datos (LDD). Además, aunque hay una orden que permite que los usuarios definan dominios nuevos, que es análoga a las órdenes de definición de tipos de los lenguajes de programación, se aplaza el tratamiento de la definición de los dominios hasta el Apartado 5.7. Por ahora sólo se considerarán los dominios que sean tipos predefinidos, como `integer`.

La instrucción `CREATE TABLE` se emplea para definir tablas nuevas[1]. Para crear la relación Alumnos se puede emplear la instrucción siguiente:

```
CREATE TABLE Alumnos ( ide      CHAR(20),
                       nombre   CHAR(30),
                       usuario  CHAR(20),
                       edad     INTEGER,
                       nota     REAL )
```

Las tuplas se insertan mediante la orden `INSERT`. Se puede insertar una sola tupla en la tabla Alumnos de la manera siguiente[2]:

```
INSERT
INTO   Alumnos   (ide, nombre, usuario, edad, nota)
VALUES (53688, 'Sánchez', 'sanchez@ii', 18, 6.4)
```

Opcionalmente se puede omitir la lista de nombres de columna de la cláusula `INTO` y relacionar los valores en el orden correspondiente, pero se considera buena práctica ser explícito acerca del nombre de las columnas.

Se pueden eliminar tuplas mediante la orden `DELETE`. Se pueden eliminar todas las tuplas de Alumnos de *nombre* igual a Sánchez empleando la orden:

```
DELETE
FROM   Alumnos A
WHERE  A.nombre = 'Sánchez'
```

Se pueden modificar los valores de las columnas de una fila ya existente mediante la orden `UPDATE`. Por ejemplo, se puede incrementar la edad y disminuir la nota del alumno con *ide* 53688:

```
UPDATE Alumnos A
SET    A.edad = A.edad + 1, A.nota = A.nota − 2
WHERE  A.ide = 53688
```

Estos ejemplos ilustran algunos puntos importantes. La cláusula `WHERE` se aplica en primer lugar y determina las filas que se van a modificar. La cláusula `SET` determina luego la manera en que se van a modificar esas filas. Si la columna que se va a modificar se emplea también para determinar el valor nuevo, el valor empleado en la expresión a la derecha del igual (=) es el valor *antiguo*, es decir, anterior a la modificación. Para ilustrar más estos puntos, considérese la siguiente variación de la consulta anterior:

```
UPDATE Alumnos A
SET    A.nota = A.nota − 0.1
WHERE  A.nota >= 6.6
```

Si esta consulta se aplica al ejemplar $A1$ de Alumnos que puede verse en la Figura 3.1, se obtiene el ejemplar mostrado en la Figura 3.3.

[1] SQL también ofrece instrucciones para eliminar tablas y para modificar las columnas asociadas a las tablas; esto se tratará en el Apartado 3.7.

[2] N. del T.: Obsérvese que se usa el punto decimal en lugar de la coma en las órdenes SQL para separar la parte fraccionaria.

ide	*nombre*	*usuario*	*edad*	*nota*
50000	Díaz	diaz@inf	19	6,4
53666	Jiménez	jimenez@inf	18	6,6
53688	Sánchez	sanchez@ii	18	6,4
53650	Sánchez	sanchez@mat	19	7,4
53831	Martínez	martinez@musica	11	3,6
53832	García	garcia@musica	12	4,0

Figura 3.3 El ejemplar $A1$ de Alumnos tras la actualización

3.2 RESTRICCIONES DE INTEGRIDAD SOBRE RELACIONES

Una base de datos sólo es tan buena como la información almacenada en ella y, por tanto, el SGBD debe ayudar a evitar la introducción de información incorrecta. Una **restricción de integridad (IC)** es una condición especificada en el esquema de la base de datos que restringe los datos que pueden almacenarse en los ejemplares de la base de datos. Si un ejemplar de la base de datos cumple todas las restricciones de integridad especificadas en el esquema de la base de datos, se trata de un ejemplar **legal**. El SGBD **hace que se cumplan** las restricciones de integridad, en el sentido de que sólo permite que se almacenen en la base de datos ejemplares legales.

Las restricciones de integridad se especifican y se hacen cumplir en momentos diferentes:

1. Cuando el DBA o el usuario final definen el esquema de la base de datos, también especifican las RI que deben cumplirse en todos los ejemplares de esa base de datos.

2. Cuando se ejecuta una aplicación, el SGBD comprueba si se produce alguna violación de las restricciones e impide las modificaciones de los datos que violen las RI especificadas. (En algunas situaciones, más que impedir la modificación, puede que el SGBD realice algunas modificaciones compensatorias en los datos para garantizar que la base de datos cumple todas las RI. En cualquier caso, no se permite que las modificaciones de la base de datos creen ejemplares que violen las RI.) Es importante especificar exactamente el momento en que se comprueba el cumplimiento de las restricciones de integridad en relación con la instrucción que provoca la modificación de los datos y la transacción de la que forma parte. Este aspecto se trata con más detalle en el Capítulo 8, tras presentar el concepto de transacción, que se introdujo en el Capítulo 1.

Se pueden especificar muchos tipos de restricciones de integridad en el modelo relacional. Ya se ha visto un ejemplo de restricción de integridad en las *restricciones de dominio* asociadas con el esquema de una relación (Apartado 3.1). En general, también se pueden especificar otros tipos de restricciones; por ejemplo, no puede haber dos alumnos con el mismo valor de *ide*. En este apartado se tratan las restricciones de integridad, aparte de las de dominio, que los DBA o los usuarios pueden especificar en el modelo relacional.

3.2.1 Restricciones de clave

Considérese la relación Alumnos y la restricción de que no puede haber dos alumnos con el mismo identificador. Esta RI es un ejemplo de restricción de clave. Una **restricción de clave** es una declaración de que un cierto subconjunto *mínimo* de los campos de una relación constituye un identificador único de cada tupla. Un conjunto de campos que identifique de manera unívoca una tupla de acuerdo con una restricción de clave se denomina **clave candidata** de esa relación; a menudo se abrevia simplemente a *clave*. En el caso de la relación Alumnos, el (conjunto de campos que sólo contiene el) campo *ide* es una clave candidata.

Examinemos más de cerca la definición anterior de clave (candidata). La definición tiene dos partes[3]:

1. Dos tuplas distintas de un ejemplar legal (un ejemplar que cumple todas las RI, incluida la restricción de clave) no pueden tener valores idénticos en todos los campos de una clave.

2. Ningún subconjunto del conjunto de campos de una clave es identificador único de una tupla.

La primera parte de la definición significa que, en *cualquier* ejemplar legal, el valor de los campos de la clave identifica de manera unívoca cada tupla de ese ejemplar. Al especificar una restricción de clave, el DBA o el usuario debe estar seguro de que esa restricción no les impida almacenar un conjunto "correcto" de tuplas. (Un comentario similar es aplicable también a la especificación de otros tipos de RI.) El concepto de "corrección" depende en este caso de la naturaleza de los datos que se vayan a almacenar. Por ejemplo, puede que varios alumnos tengan el mismo nombre, aunque cada uno tenga un identificador único. Si se declara que el campo *nombre* es una clave, el SGBD no permitirá que la relación Alumnos contenga dos tuplas que describan a alumnos diferentes con el mismo nombre.

La segunda parte de la definición significa, por ejemplo, que el conjunto de campos {*ide, nombre*} no es una clave de Alumnos, ya que este conjunto contiene a su vez a la clave {*ide*}. El conjunto {*ide, nombre*} es un ejemplo de **superclave**, que es un conjunto de campos que contiene una clave.

Examinemos de nuevo el ejemplar de la relación Alumnos de la Figura 3.1. Obsérvese que dos filas diferentes cualesquiera tienen siempre valores diferentes de *ide*; *ide* es una clave e identifica de manera unívoca a cada tupla. No obstante, esto no es válido para los campos que no constituyen la clave. Por ejemplo, la relación contiene dos filas con *Sánchez* en el campo *nombre*.

Obsérvese que se garantiza que cada relación tenga una clave. Dado que una relación es un conjunto de tuplas, el conjunto de *todos* los campos es siempre una superclave. Si se cumplen otras restricciones, puede que algún subconjunto de los campos forme una clave pero, si no, el conjunto de todos los campos será la clave.

Cada relación puede tener varias claves candidatas. Por ejemplo, los campos *usuario* y *edad* de la relación Alumnos también pueden, considerados en conjunto, identificar de manera unívoca a los alumnos. Es decir, {*usuario, edad*} es también una clave. Puede parecer que

[3]Se abusa bastante del término *clave*. En el contexto de los métodos de acceso se habla de *claves de búsqueda*, que son bastante diferentes.

usuario es una clave, ya que no hay dos filas del ejemplar de ejemplo que tengan el mismo valor de *usuario*. Sin embargo, la clave debe identificar las tuplas de manera unívoca en todos los ejemplares legales posibles de la relación. Al declarar que {*usuario, edad*} es una clave, el usuario declara que dos alumnos pueden tener el mismo usuario o la misma edad, pero no las dos cosas a la vez.

De entre todas las claves candidatas disponibles, un diseñador de bases de datos puede identificar una clave **principal**. De manera intuitiva, se puede hacer referencia a cada tupla desde cualquier punto de la base de datos mediante el almacenamiento del valor de los campos de su clave principal. Por ejemplo, se puede hacer referencia a una tupla de Alumnos almacenando su valor de *ide*. Como consecuencia de que se haga referencia de esta manera a las tuplas de los alumnos, es frecuente que se tenga acceso a las tuplas mediante la especificación de su valor de *ide*. En principio, se puede emplear cualquier clave para hacer referencia a una tupla dada además de con la clave principal. Sin embargo, es preferible el empleo de la clave principal, ya que es lo que espera el SGBD que se haga —he aquí la trascendencia de la designación de una clave candidata dada como clave principal— y para lo que realiza la optimización. Por ejemplo, puede que el SGBD cree un índice con los campos de la clave principal como clave de búsqueda para hacer eficiente la recuperación de una tupla a partir del valor de su clave principal. La idea de hacer referencia a las tuplas se desarrolla más a fondo en el apartado siguiente.

Especificación de restricciones de clave en SQL

En SQL se puede declarar que un subconjunto de las columnas de una tabla constituye una clave mediante la restricción `UNIQUE`. Se puede declarar como máximo que una de esas claves candidatas es la *clave principal*, mediante la restricción `PRIMARY KEY`. (SQL no exige que se declaren esas restricciones para las tablas.)

Volvamos a la definición de nuestro ejemplo de tabla y especifiquemos la información de la clave:

```
CREATE TABLE Alumnos ( ide       CHAR(20),
                       nombre    CHAR(30),
                       usuario   CHAR(20),
                       edad      INTEGER,
                       nota      REAL,
                       UNIQUE (nombre, edad),
                       CONSTRAINT ClaveAlumnos PRIMARY KEY (ide) )
```

Esta definición indica que *ide* es la clave principal y que la combinación de *nombre* y de *edad* también es una clave. La definición de la clave primaria también ilustra la manera en que se puede denominar una restricción anteponiéndole `CONSTRAINT` *nombre-restricción*. Si se viola la restricción, se devuelve el nombre de la restricción y se puede emplear para identificar el error.

3.2.2 Restricciones de clave externa

A veces la información almacenada en una relación está vinculada con la información almacenada en otra. Si se modifica una de las relaciones hay que comprobar la otra y, quizás, modificarla para hacer que los datos sigan siendo consistentes. Hay que especificar una RI que implique a las dos relaciones si esas comprobaciones debe hacerlas el SGBD. La RI que implica a dos relaciones más frecuente es la restricción de *clave externa*.

Supóngase que, además de Alumnos, tenemos una segunda relación:

Matriculado(*idalum:* `string`, *ida:* `string`, *nota:* `string`)

Para garantizar que sólo se pueden matricular de las asignaturas auténticos alumnos, cualquier valor que aparezca en el campo *idalum* de un ejemplar de la relación Matriculado debe aparecer también en el campo *ide* de alguna tupla de la relación Alumnos. El campo *idalum* de Matriculado se denomina **clave externa** y **hace referencia** a Alumnos. La clave externa de la relación que hace la referencia (Matriculado, en este ejemplo) debe coincidir con la clave principal de la relación a la que se hace referencia (Alumnos); es decir, debe tener el mismo número de columnas y tipos de datos compatibles, aunque el nombre de las columnas puede ser diferente.

Esta restricción se ilustra en la Figura 3.4. Como muestra la figura, puede que haya algunas tuplas de Alumnos a las que no se haga referencia desde Matriculado (por ejemplo, el alumno con *ide=50000*). Sin embargo, todos los valores de *idalum* que aparecen en el ejemplar de la tabla Matriculado aparecen en la columna de la clave principal de la tabla Alumnos.

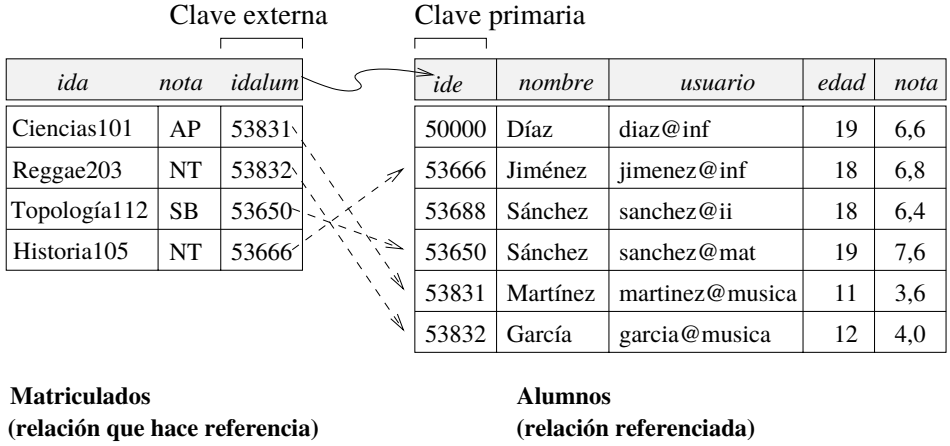


Figura 3.4 Integridad referencial

Si se intenta insertar la tupla ⟨*55555, Art104, SB*⟩ en $M1$ se viola la RI, ya que no hay ninguna tupla de $A1$ con *ide* 55555; el sistema de bases de datos debe rechazar esa inserción. De manera parecida, si se elimina la tupla ⟨*53666, Jiménez, jimenez@inf, 18, 6,8*⟩ de $A1$ se viola la restricción de clave externa, ya que la tupla ⟨*53666, Historia105, NT*⟩ de $M1$ contiene el valor de *idalum* 53666, la *ide* de la tupla Alumno eliminada. El SGBD debe impedir la eliminación o, quizás, eliminar también la tupla de Matriculado que hace referencia a la tupla

eliminada de Alumnos. Las restricciones de clave externa y su efecto en las actualizaciones se tratan en el Apartado 3.3.

Finalmente, hay que destacar que las claves externas pueden hacer referencia a la misma relación en que se hallan. Por ejemplo, se puede ampliar la relación Alumnos con una columna denominada *compañero* y declarar que esa columna es una clave externa que hace referencia a Alumnos. De manera intuitiva, cada alumno podrá tener un compañero y el campo *compañero* contiene el *ide* de éste. No cabe duda de que el lector observador se preguntará, "¿Qué ocurre si un alumno no tiene (todavía) un compañero?" Esta situación se resuelve en SQL mediante un valor especial denominado **null** (nulo). El uso de *null* en un campo de una tupla indica que el valor de ese campo es desconocido o no se puede aplicar (por ejemplo, no se conoce todavía al compañero o no hay ninguno). La aparición de *null* en un campo de clave externa no viola la restricción de clave externa. Sin embargo, no se permite que aparezcan valores *null* en los campos de la clave principal (ya que los campos de la clave principal se emplean para identificar la tupla de manera unívoca). Los valores *null* se tratan con mayor profundidad en el Capítulo 5.

Especificación de restricciones de clave externa en SQL

Definamos Matriculado(*idalum:* `string`, *ida:* `string`, *nota:* `string`):

```
CREATE TABLE Matriculado ( idalum CHAR(20),
                           ida    CHAR(20),
                           nota   CHAR(10),
                           PRIMARY KEY (idalum, ida),
                           FOREIGN KEY (idalum) REFERENCES Alumnos )
```

La restricción de clave externa afirma que todos los valores de *idalum* de Matriculado deben aparecer también en Alumnos, es decir, *idalum* de Matriculado es una clave externa que hace referencia a Alumnos. Más concretamente, cada valor de *idalum* de Matriculado debe aparecer como valor del campo de clave principal, *ide*, de Alumnos. Por cierto, la restricción de clave principal de Matriculado afirma que cada alumno tiene exactamente una nota por cada asignatura en la que se haya matriculado. Si se desea registrar más de una, se debe modificar la restricción de clave principal.

3.2.3 Restricciones generales

Las restricciones de dominio, de clave principal y de clave externa se consideran una parte fundamental del modelo relacional de datos y se les presta especial atención en la mayor parte de los sistemas comerciales. A veces, sin embargo, resulta necesario especificar restricciones más generales.

Por ejemplo, puede que se necesite que la edad de los alumnos caiga dentro de un determinado rango de valores; dada esa especificación de RI, el SGBD rechaza las inserciones y las actualizaciones que la violen. Esto resulta muy útil para evitar errores de introducción de datos. Si se especifica que todos los alumnos deben tener, como mínimo, 16 años de edad, el ejemplar de Alumnos mostrado en la Figura 3.1 es ilegal, ya que dos de los alumnos son más

jóvenes. Si se impide la inserción de esas tuplas se tiene un ejemplar legal, como puede verse en la Figura 3.5.

ide	*nombre*	*usuario*	*edad*	*nota*
53666	Jiménez	jimenez@inf	18	6,8
53688	Sánchez	sanchez@ii	18	6,4
53650	Sánchez	sanchez@mat	19	7,6

Figura 3.5 El ejemplar $A2$ de la relación Alumnos

La RI de que los alumnos deben tener más de 16 años se puede considerar una restricción de dominio ampliada, ya que básicamente se define el conjunto de valores admisibles de *edad* más estrictamente de lo posible mediante el mero empleo de un dominio estándar como `integer`. En general, sin embargo, se pueden especificar restricciones que van más allá de las de dominio, clave o clave externa. Por ejemplo, se puede exigir que todos los alumnos de edad superior a 18 tenga una nota superior a 6.

Los sistemas de bases de datos relacionales actuales permiten restricciones así de generales en forma de *restricciones de tabla* y *asertos*. Las restricciones de tabla se asocian con una sola tabla y se comprueban siempre que ésta se modifica. Por el contrario, los asertos implican a varias tablas y se comprueban siempre que se modifica alguna de ellas. Tanto las restricciones de tabla como los asertos pueden emplear toda la potencia de las consultas de SQL para especificar la restricción deseada. El soporte en SQL de las *restricciones de tabla* y de los *asertos* se trata en el Apartado 5.7, ya que entender bien su potencia exige una buena comprensión de las capacidades de consulta de SQL.

3.3 CUMPLIMIENTO DE LAS RESTRICCIONES DE INTEGRIDAD

Como ya se ha observado, las RI se especifican al crear la relación y se hacen cumplir cuando ésta se modifica. El impacto de las restricciones de dominio, `PRIMARY KEY` y `UNIQUE` es inmediato: si alguna orden de inserción, eliminación o actualización provoca una violación, se rechaza. Todas las posibles violaciones de las RI se suelen comprobar al final de la ejecución de cada instrucción de SQL, aunque esto se puede *diferir* hasta el final de la transacción que ejecuta esa instrucción, como se verá en el Apartado 3.3.1.

Considérese el ejemplar $A1$ de Alumnos que puede verse en la Figura 3.1. La siguiente inserción viola la restricción de clave principal, pues ya hay una tupla con la *ide* 53688, y el SGBD la rechazará:

```
INSERT
INTO   Alumnos    (ide, nombre, usuario, edad, nota)
VALUES (53688, 'Miguel', 'miguel@ii', 17, 6.8)
```

La siguiente inserción viola la restricción de que la clave principal no puede contener valores *nulos*:

```
INSERT
```

```
INTO    Alumnos    (ide, nombre, usuario, edad, nota)
VALUES  (null, 'Miguel', 'miguel@ii', 17, 6.8)
```

Por supuesto, surgirá un problema parecido siempre que se intente insertar una tupla con valores de algún campo que no se hallen en el dominio asociado a ese campo, es decir, siempre que se viole una restricción de dominio. La eliminación no provoca violaciones de dominio, de clave principal o de unicidad. Sin embargo, las actualizaciones pueden provocar violaciones, de modo similar a las inserciones:

```
UPDATE  Alumnos A
SET     A.ide = 50000
WHERE   A.ide = 53688
```

Esta actualización viola la restricción de clave principal, pues ya hay una tupla con *ide* 50000.

El efecto de las restricciones de clave externa es más complejo, ya que a veces SQL intenta rectificar las violaciones de clave externa en lugar de limitarse a rechazar las modificaciones. Se tratarán las **etapas de cumplimiento de la integridad referencial** seguidas por el SGBD en términos de las tablas Matriculado y Alumnos de nuestro ejemplo, con la restricción de clave externa de que Matriculado.*ide* es una referencia para (la clave primaria de) Alumnos.

Además del ejemplar $A1$ de Alumnos, considérese el ejemplar de Matriculado que puede verse en la Figura 3.4. Las eliminaciones de tuplas de Matriculado no violan la integridad referencial, pero las inserciones de tuplas de Matriculado podrían hacerlo. La inserción siguiente es ilegal porque no hay ninguna tupla de Alumnos con *ide* 51111:

```
INSERT
INTO    Matriculado (ida, nota, idalum)
VALUES  ('Inglés101', 'NT', 51111)
```

Por otro lado, las inserciones de tuplas de Alumnos no violan la integridad referencial, y las eliminaciones de tuplas de Alumnos sí podrían hacerlo. Además, tanto las actualizaciones de Matriculado como las de Alumnos que modifiquen el valor de *idalum* (respectivamente, *ide*) podrían acabar violando la integridad referencial.

SQL ofrece varios métodos alternativos para tratar las violaciones de las claves externas. Se deben considerar tres aspectos básicos:

1. *¿Qué hacer si se inserta una fila de Matriculado con un valor de la columna* idalum *que no aparezca en ninguna fila de la tabla Alumnos?*

   En este caso, simplemente se rechaza la orden `INSERT`.

2. *¿Qué hacer si se elimina una fila de Alumnos?*

   Las opciones son:

   - Eliminar todas las filas de Matriculado que hagan referencia a la fila de Alumnos eliminada.
   - Impedir la eliminación de la fila de Alumnos si alguna fila de Matriculado hace referencia a ella.

- Asignar la columna *idalum* a la *ide* de algún alumno (existente) "predeterminado", para cada fila de Matriculado que haga referencia a la fila de Alumnos eliminada.
- Para cada fila de Matriculado que haga referencia a ella, definir la columna *idalum* como *null*. En nuestro ejemplo, esta opción entra en conflicto con el hecho de que *idalum* forma parte de la clave principal de Matriculado y, por tanto, no se puede definir como *null*. Por tanto, en este ejemplo quedamos restringidos a las tres primeras opciones, aunque esta cuarta opción (definir la clave externa como *null*), en general, está disponible.

3. *¿Qué hacer si el valor de la clave principal de una fila de Alumnos se actualiza?*

   Las opciones son parecidas a las del caso anterior.

SQL permite escoger cualquiera de las cuatro opciones de `DELETE` y de `UPDATE`. Por ejemplo, se puede especificar que, cuando se *elimine* una fila de Alumnos, se eliminen también todas las filas de Matriculado que hagan referencia a ella, pero que cuando se *modifique* la columna *ide* de Alumnos se rechace esa actualización si alguna fila de Matriculado hace referencia a la fila modificada de Alumnos:

```
CREATE TABLE Matriculado (  idalum CHAR(20),
                            ida    CHAR(20),
                            nota   CHAR(10),
                            PRIMARY KEY (idalum, ida),
                            FOREIGN KEY (idalum) REFERENCES Alumnos
                                        ON DELETE CASCADE
                                        ON UPDATE NO ACTION )
```

Las opciones se especifican como parte de la declaración de clave externa. La opción predeterminada es `NO ACTION`, que significa que la acción (`DELETE` o `UPDATE`) se debe rechazar. Por tanto, la cláusula `ON UPDATE` del ejemplo se puede omitir con idéntico efecto. La palabra clave `CASCADE` indica que, si se elimina alguna fila de Alumnos, también se eliminarán todas las filas de Matriculado que hagan referencia a ella. Si la cláusula `UPDATE` especificara `CASCADE` y la columna *ide* de alguna fila de Alumnos se actualizase, esa actualización también se llevaría a cabo en cada fila de Matriculado que hiciera referencia a la fila de Alumnos actualizada.

Si se elimina una fila de Alumnos se puede cambiar la matrícula a un alumno "predeterminado" mediante `ON DELETE SET DEFAULT`. El alumno predeterminado se especifica como parte de la definición del campo *ide* en Matriculado; por ejemplo, *ide* `CHAR(20) DEFAULT` *'53666'*. Aunque la especificación de un valor predeterminado resulta adecuada en algunas situaciones (por ejemplo, un proveedor de repuestos predeterminado si un proveedor dado cierra), no resulta realmente adecuado cambiar las matrículas a un alumno predeterminado. La solución correcta en este ejemplo es eliminar también todas las tuplas de matrícula del alumno eliminado (es decir, `CASCADE`) o rechazar la actualización.

SQL también permite el empleo de valores *null* como valor predeterminado especificando `ON DELETE SET NULL`.

3.3.1 Transacciones y restricciones

Como se vio en el Capítulo 1, los programas que se ejecutan contra las bases de datos se denominan transacciones, y pueden contener varias instrucciones (consultas, inserciones, actualizaciones, etcétera) que tengan acceso a la base de datos. Si (la ejecución de) alguna de las instrucciones viola una restricción de integridad, ¿debe el SGBD detectarlo inmediatamente o se deben comprobar todas las restricciones conjuntamente justo antes de que se complete la transacción?

De manera predeterminada, cada restricción se comprueba al final de todas las instrucciones de SQL que puedan provocar una violación y, si ésta se produce, la instrucción se rechaza. A veces este enfoque resulta demasiado inflexible. Considérense las siguientes variantes de las relaciones Alumnos y Asignaturas; se exige que todos los alumnos tengan una asignatura avanzada y que cada asignatura tenga un alumno, que es alguno de los alumnos.

```
CREATE TABLE Alumnos (    ide        CHAR(20),
                          nombre     CHAR(30),
                          usuario    CHAR(20),
                          edad       INTEGER,
                          avanzada   CHAR(10) NOT NULL,
                          nota       REAL )
                          PRIMARY KEY (ide),
                          FOREIGN KEY (avanzada) REFERENCES Asignaturas (ida))

CREATE TABLE Asignaturas ( ida       CHAR(10),
                           nombrea   CHAR(10),
                           créditos  INTEGER,
                           alumno    CHAR(20) NOT NULL,
                           PRIMARY KEY (ida)
                           FOREIGN KEY (alumno) REFERENCES Alumnos (ide))
```

Siempre que se inserta una tupla de Alumnos se realiza una comprobación para ver si la asignatura avanzada se halla en la relación Asignaturas, y siempre que se inserta una tupla de Asignaturas se realiza una comprobación para ver si el alumno se halla en la relación Alumnos. ¿Cómo se puede insertar la primera tupla de asignatura o de alumno? No se puede insertar una sin la otra. La única manera de lograrlo es **diferir** la comprobación de la restricción, que normalmente se llevaría a cabo al final de la instrucción `INSERT`.

SQL permite que las restricciones se hallen en modo `DEFERRED` o en modo `IMMEDIATE`.

```
SET CONSTRAINT Restricción DEFERRED
```

Las restricciones en modo diferido se comprueban en el momento del compromiso. En este ejemplo se puede declarar que tanto la restricción de clave externa de Alumnos como la de Asignaturas están en modo diferido. Luego se puede insertar un alumno sin una asignatura avanzada (lo que hace que la base de datos sea temporalmente inconsistente), insertar la asignatura avanzada (lo cual restaura la consistencia) y luego comprometer la transacción y comprobar que se satisfacen las dos restricciones.

3.4 CONSULTAS DE DATOS RELACIONALES

Una **consulta a una base de datos relacional** (consulta, para abreviar) es una pregunta sobre los datos, y la respuesta consiste en una nueva relación que contiene el resultado. Por ejemplo, puede que se desee averiguar todos los alumnos menores de 18 años o todos los alumnos matriculados en Reggae203. Un **lenguaje de consulta** es un lenguaje especializado para la escritura de consultas.

SQL es el lenguaje de consulta comercial más popular para los SGBD relacionales. A continuación se ofrecen algunos ejemplos de SQL que ilustran la facilidad con que se pueden formular consultas a las relaciones. Considérese el ejemplar de la relación Alumnos mostrado en la Figura 3.1. Se pueden recuperar las filas correspondientes a los alumnos que son menores de 18 años con la siguiente consulta de SQL:

```
SELECT *
FROM   Alumnos A
WHERE  A.edad < 18
```

El símbolo * indica que se conservarán en el resultado todos los campos de las tuplas seleccionadas. Se puede pensar en A como una variable que toma el valor de cada tupla de Alumnos, una tras otra. La condición *A.edad < 18* de la cláusula `WHERE` especifica que sólo se desea seleccionar las tuplas en las que el campo *edad* tenga un valor menor de 18. El resultado de la evaluación de esta consulta se muestra en la Figura 3.6.

ide	*nombre*	*usuario*	*edad*	*nota*
53831	Martínez	martinez@musica	11	7,2
53832	García	garcia@musica	12	8,0

Figura 3.6 Alumnos con *edad* < 18 en el ejemplar $A1$

Este ejemplo muestra que el dominio de cada campo restringe las operaciones que están permitidas sobre los valores de ese campo, además de restringir los valores que pueden aparecer en ese campo. La condición *A.edad < 18* implica la comparación aritmética de un valor de *edad* con un número entero, y es admisible porque el dominio de *edad* es el conjunto de los números enteros. Por otro lado, una condición como *A.edad = A.ide* no tiene sentido, ya que compara un valor entero con un valor de cadena de caracteres, y por definición estas comparaciones fallan en SQL; las consultas que contengan esa condición no producirán ninguna tupla como respuesta.

Además de seleccionar un subconjunto de tuplas, cada consulta puede extraer un subconjunto de los campos de las tuplas seleccionadas. Se pueden calcular los nombres y los usuarios de los alumnos menores de 18 años con la consulta siguiente:

```
SELECT A.nombre, A.usuario
FROM   Alumnos A
WHERE  A.edad < 18
```

La Figura 3.7 muestra la respuesta a esta consulta; se obtiene aplicando la selección al ejemplar $A1$ de Alumnos (para obtener la relación que aparece en la Figura 3.6), seguida de

la eliminación de los campos no deseados. Obsérvese que el orden en que se llevan a cabo estas operaciones es importante —si se eliminan en primer lugar los campos no deseados no se puede comprobar la condición $A.edad < 18$, que implica a uno de esos campos—.

nombre	*usuario*
Martínez	martinez@musica
García	garcia@musica

Figura 3.7 Nombre y usuario de los alumnos menores de 18 años

También se puede combinar la información de las relaciones Alumnos y Matriculado. Si se desea obtener el nombre de todos los alumnos que obtuvieron un sobresaliente y el identificador de la asignatura en que lo lograron, se puede escribir la consulta siguiente:

```
SELECT A.nombre, M.ida
FROM   Alumnos A, Matriculado M
WHERE  A.ide = M.idalum AND M.nota = 'SB'
```

Esta consulta puede entenderse de la manera siguiente: "Si hay una tupla de Alumnos A y una tupla de Matriculado M tales que A.ide = M.idalum (de modo que A describe al alumno matriculado de M) y M.nota = 'SB', hay que escribir el nombre del alumno y la id de la asignatura." Cuando esta consulta se evalúa sobre los ejemplares de Alumnos y Matriculado de la Figura 3.4, se devuelve una sola tupla, $\langle$*Sánchez, Topología112*$\rangle$.

Las consultas relacionales y SQL se tratan con más detalle en capítulos posteriores.

3.5 DISEÑO LÓGICO DE BASES DE DATOS: DEL MODELO ER AL RELACIONAL

El modelo ER resulta conveniente para representar un diseño inicial de alto nivel de la base de datos. Dado un diagrama ER que describe una base de datos, se sigue el enfoque estándar para generar un esquema relacional de la base de datos que se aproxime lo más posible al diseño ER. (La traducción es aproximada debido a que no se pueden capturar todas las restricciones implícitas en el diseño ER mediante SQL, a menos que se empleen ciertas restricciones de SQL que resultan difíciles de comprobar.) A continuación se describirá el modo de traducir cada diagrama ER en un conjunto de tablas con restricciones asociadas, es decir, el esquema de una base de datos relacional.

3.5.1 De los conjuntos de entidades a las tablas

Cada conjunto de entidades se asigna a una relación de una manera directa: cada atributo de la entidad se convierte en atributo de una tabla. Obsérvese que se conocen tanto el dominio de cada atributo como la clave (principal) del conjunto de entidades.

Considérese el conjunto de entidades Empleados con los atributos *dni, nombre* y *plaza* que puede verse en la Figura 3.8.

Un posible ejemplar del conjunto de entidades Empleados, que contiene tres entidades Empleados, puede verse en la Figura 3.9 en formato tabular.

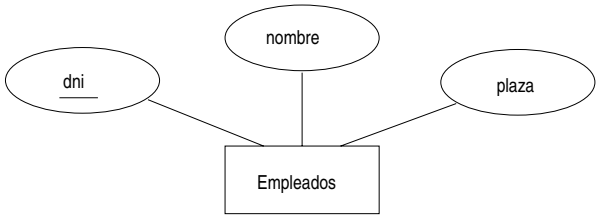


Figura 3.8 El conjunto de entidades Empleados

dni	*nombre*	*plaza*
123.223.666	Avelino	48
231.315.368	Serna	22
131.243.650	Soria	35

Figura 3.9 Un ejemplar del conjunto de entidades Empleados

La siguiente instrucción de SQL captura la información anterior, incluidas las restricciones de dominio y la información sobre las claves:

```
CREATE TABLE Empleados ( dni      CHAR(11),
                         nombre   CHAR(30),
                         plaza    INTEGER,
                         PRIMARY KEY (dni) )
```

3.5.2 De los conjuntos de relaciones (sin restricciones) a las tablas

Cada conjunto de relaciones, al igual que los conjuntos de entidades, se asigna a una relación del modelo relacional. Se comienza por considerar conjuntos de relaciones sin restricciones de clave ni de participación y en apartados posteriores se estudia la manera de manejar esas restricciones. Para representar una relación hay que poder identificar cada entidad participante y asignar valores a los atributos descriptivos de esa relación. Así, entre los atributos de la relación están:

- Los atributos de la clave principal de cada conjunto de entidades participante, como los campos que forman clave externa.
- Los atributos descriptivos del conjunto de relaciones.

El conjunto de atributos no descriptivos es una superclave de la relación. Si no existen restricciones de clave (véase el Apartado 2.4.1), este conjunto de atributos es una clave candidata.

Considérese el conjunto de relaciones Trabaja_en2 que puede verse en la Figura 3.10. Cada departamento tiene despachos en varias ubicaciones y se desea registrar las ubicaciones en que trabaja cada empleado.

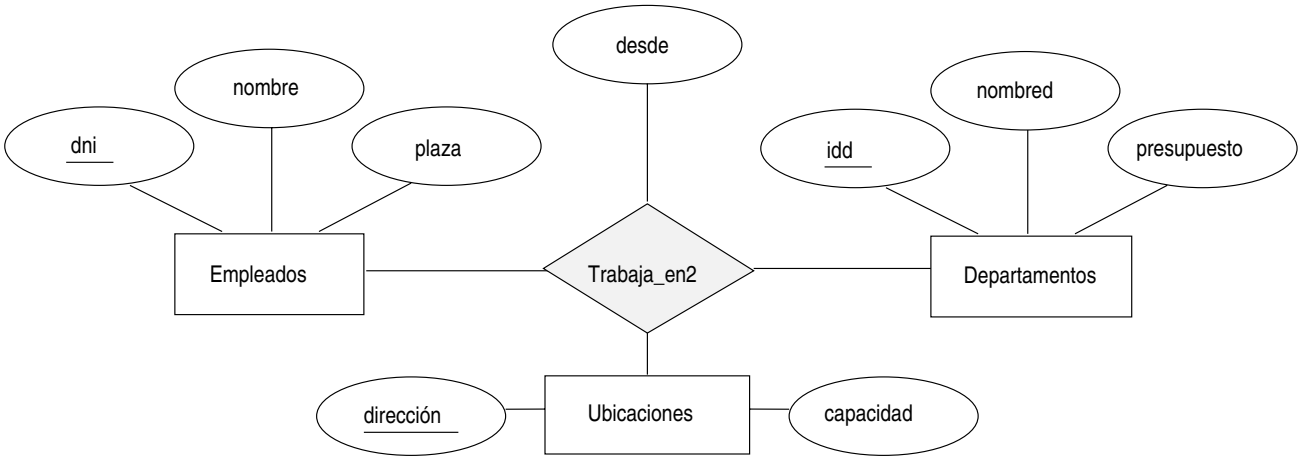


Figura 3.10 Un conjunto de relaciones ternarias

Toda la información disponible sobre la tabla Trabaja_en2 se captura mediante la siguiente definición de SQL:

```
CREATE TABLE Trabaja_en2 ( dni       CHAR(11),
                           idd       INTEGER,
                           dirección CHAR(20),
                           desde     DATE,
                           PRIMARY KEY (dni, idd, dirección),
                           FOREIGN KEY (dni) REFERENCES Empleados,
                           FOREIGN KEY (dirección) REFERENCES Ubicaciones,
                           FOREIGN KEY (idd) REFERENCES Departamentos )
```

Obsérvese que los campos *dirección*, *idd* y *dni* no pueden adoptar valores *null*. Como estos campos forman parte de la clave principal de Trabaja_en2, queda implícita una restricción `NOT NULL` para cada uno de estos campos. Esta restricción garantiza que esos campos identifiquen de manera unívoca a un departamento, un empleado y una ubicación de cada tupla de Trabaja_en2. También se puede especificar que una acción determinada es deseable cuando se elimine una tupla de Empleados, Departamentos o Ubicaciones a la que se haga referencia, como se explica en el tratamiento de las restricciones de integridad en el Apartado 3.2. En este capítulo se supone que la acción predeterminada resulta adecuada, salvo en las situaciones en las que la semántica del diagrama ER exija alguna acción diferente.

Finalmente, considérese el conjunto de relaciones Informa_a que puede verse en la Figura 3.11. Los indicadores de papeles *supervisor* y *subordinado* se emplean para crear nombres de campo significativos en la instrucción `CREATE` para la tabla Informa_a:

```
CREATE TABLE Informa_a (
        dni_supervisor   CHAR(11),
        dni_subordinado CHAR(11),
        PRIMARY KEY (dni_supervisor, dni_subordinado),
        FOREIGN KEY (dni_supervisor) REFERENCES Empleados(dni),
        FOREIGN KEY (dni_subordinado) REFERENCES Empleados(dni) )
```

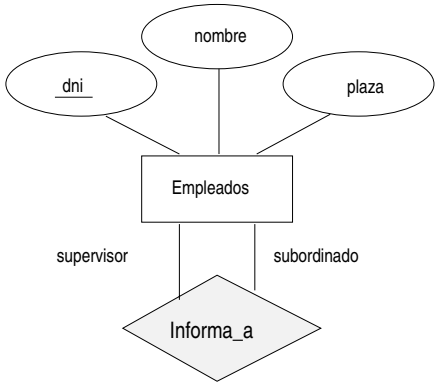


Figura 3.11 El conjunto de relaciones Informa_a

Obsérvese que hay que nombrar de manera explícita el campo de Empleados al que se hace referencia porque el nombre del campo no coincide con el nombre de los campos que hacen la referencia.

3.5.3 Traducción de conjuntos de relaciones con restricciones de clave

Si un conjunto de relaciones implica a n conjuntos de entidades y m de ellos están unidos por flechas en el diagrama ER, la clave de cualquiera de esos m conjuntos de entidades constituye una clave de la relación a la que se asigne ese conjunto de relaciones. Por tanto, se tienen m claves candidatas, una de las cuales debe designarse como clave principal. La traducción estudiada en el Apartado 2.3 de conjuntos de relaciones a relaciones se puede emplear en presencia de restricciones de clave, teniendo en cuenta esta consideración sobre las claves.

Considérese el conjunto de relaciones Dirige de la Figura 3.12. La tabla correspondiente

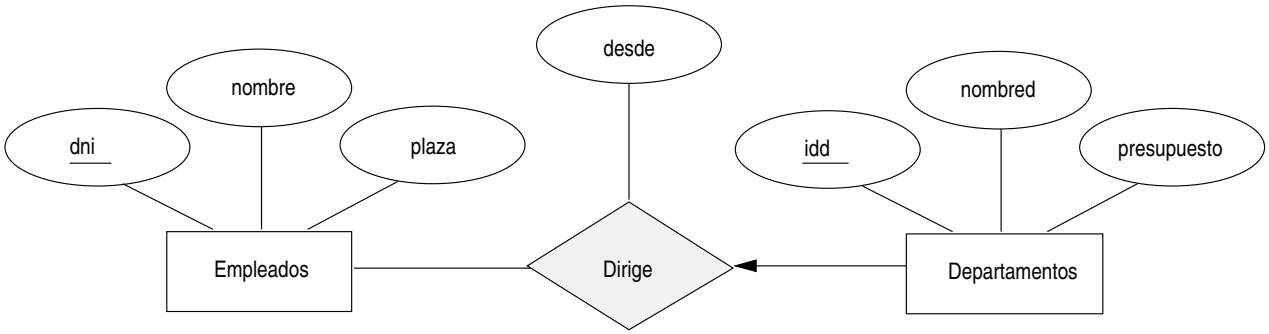


Figura 3.12 Restricción de clave para Dirige

a Dirige tiene los atributos *dni, idd* y *desde*. Sin embargo, como cada departamento tiene, como máximo, un encargado, no puede haber dos tuplas que tengan el mismo valor de *idd* y no coincidan en el valor de *dni*. Una consecuencia de esta observación es que *idd* es, en

sí misma, una clave de Dirige; en realidad, el conjunto *idd, dni* no es una clave (porque no es mínimo). La relación Dirige se puede definir mediante la siguiente instrucción de SQL:

```
CREATE TABLE Dirige ( dni      CHAR(11),
                      idd      INTEGER,
                      desde    DATE,
                      PRIMARY KEY (idd),
                      FOREIGN KEY (dni) REFERENCES Empleados,
                      FOREIGN KEY (idd) REFERENCES Departamentos )
```

Un segundo enfoque de la traducción de los conjuntos de relaciones con restricciones de clave suele resultar más adecuado, ya que evita la creación de una tabla diferente para el conjunto de relaciones. La idea es incluir la información sobre el conjunto de relaciones en la tabla correspondiente al conjunto de entidades con la clave, aprovechando la restricción de integridad. En el ejemplo Dirige, como cada departamento tiene como máximo un encargado, se pueden añadir los campos de la clave de la tupla Empleados que denotan al encargado y el atributo *desde* a la tupla Departamentos.

Este enfoque elimina la necesidad de una relación Dirige independiente, y las consultas que buscan encargados de departamento se pueden responder sin combinar información de dos relaciones. El único inconveniente de este enfoque es que se puede desperdiciar espacio si varios departamentos carecen de encargado. En ese caso, habrá que rellenar los campos añadidos con valores *null*. La primera traducción (que emplea una tabla independiente para Dirige) evita esa ineficiencia, pero algunas consultas importantes obligan a combinar información de dos relaciones, lo que puede suponer una operación lenta.

La siguiente instrucción de SQL, que define una relación Dept_Enc, que captura la información tanto de Departamento como de Dirige, ilustra el segundo enfoque de la traducción de conjuntos de relaciones con restricciones de clave:

```
CREATE TABLE Dept_Enc ( idd          INTEGER,
                        nombred      CHAR(20),
                        presupuesto  REAL,
                        dni          CHAR(11),
                        desde        DATE,
                        PRIMARY KEY (idd),
                        FOREIGN KEY (dni) REFERENCES Empleados )
```

Obsérvese que *dni* puede adoptar valores *null*.

Esta idea puede ampliarse para tratar con los conjuntos de relaciones que impliquen a más de dos conjuntos de entidades. En general, si un conjunto de relaciones implica a n conjuntos de entidades y m de ellos están unidos por flechas en el diagrama ER, la relación correspondiente a cualquiera de los m conjuntos se puede ampliar para que capture la relación.

Se tratarán los méritos relativos de los dos enfoques de la traducción tras considerar la manera de traducir en tablas los conjuntos de relaciones con restricciones de participación.

3.5.4 Traducción de los conjuntos de relaciones con restricciones de participación

Considérese el diagrama ER de la Figura 3.13, que muestra dos conjuntos de relaciones, Dirige y Trabaja_en.

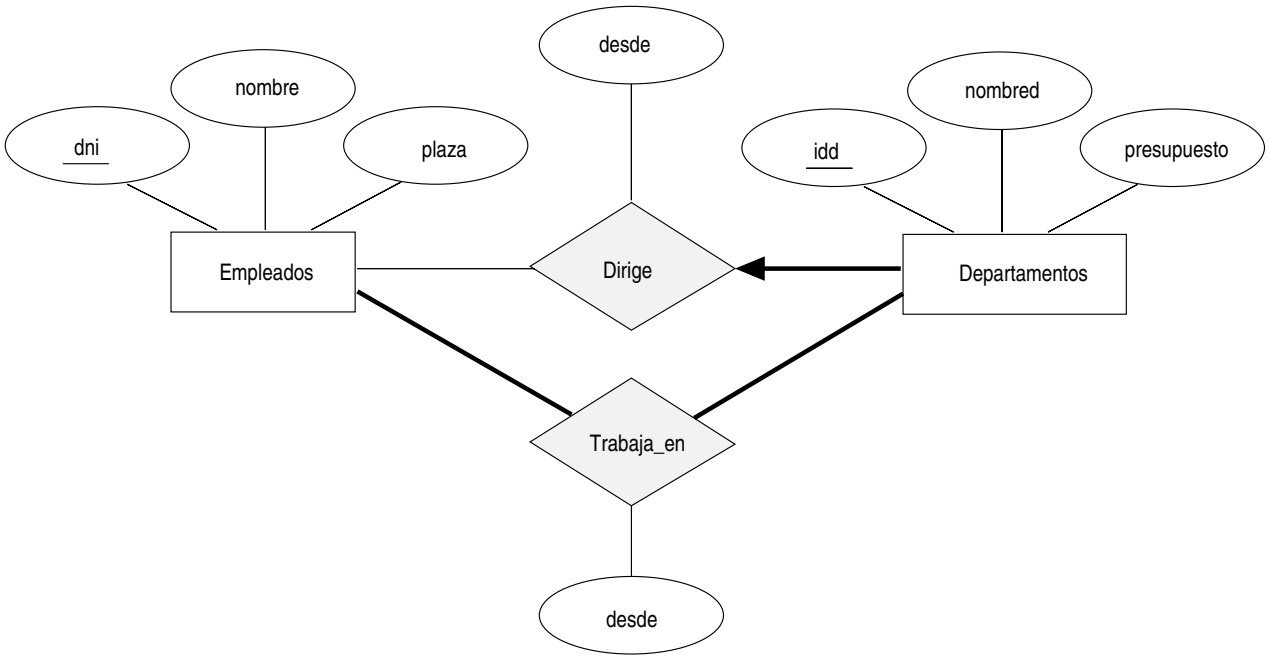


Figura 3.13 Dirige y Trabaja_en

Se exige que todos los departamentos tengan un encargado, debido a la restricción de participación, y como máximo un encargado, debido a la restricción de clave. La siguiente instrucción SQL refleja el segundo enfoque de traducción tratado en el Apartado 3.5.3, y emplea la restricción de clave:

```
CREATE TABLE Dept_Enc ( idd          INTEGER,
                        nombred      CHAR(20),
                        presupuesto  REAL,
                        dni          CHAR(11) NOT NULL,
                        desde        DATE,
                        PRIMARY KEY (idd),
                        FOREIGN KEY (dni) REFERENCES Empleados
                                ON DELETE NO ACTION )
```

También captura la restricción de participación de que cada departamento debe tener un encargado: como *dni* no puede adoptar valores *null*, cada tupla de Dept_Enc identifica a una tupla de Empleados (la del encargado). La especificación `NO ACTION`, que es la predeterminada y no hace falta especificarla explícitamente, garantiza que no se puedan borrar tuplas de Empleados mientras apunte a ellas alguna tupla de Dept_Enc. Si se desea eliminar alguna de esas tuplas de Empleados hay que modificar antes la tupla de Dept_Enc para que tenga a otro

empleado como encargado. (Se podría habar especificado `CASCADE` en lugar de `NO ACTION`, pero parece un poco radical eliminar toda la información relativa a un departamento sólo porque se haya despedido a su encargado.)

La restricción de que cada departamento deba tener un encargado no se puede capturar empleando el primer enfoque de la traducción estudiado en el Apartado 3.5.3. (Examínese la definición de Dirige y medítese sobre el efecto que tendría añadir restricciones `NOT NULL` a los campos *dni* e *idd*. *Sugerencia*: la restricción evitaría el despido del encargado, pero no garantiza que se nombre un encargado para cada departamento.) Esta situación es un fuerte argumento a favor del empleo del segundo enfoque para las relaciones de uno a varios como Dirige, especialmente cuando el conjunto de entidades con la restricción de clave tiene también una restricción de participación total.

Desafortunadamente hay muchas restricciones de participación que no se pueden capturar mediante SQL, a menos que se empleen *restricciones de tabla* o *asertos*. Las restricciones de tabla y los asertos se pueden especificar empleando toda la potencia del lenguaje de consulta SQL (como se ve en el Apartado 5.7) y son muy expresivos pero, también, muy costosos de comprobar y de hacer cumplir. Por ejemplo, no se pueden hacer cumplir las restricciones de participación de la relación Trabaja_en sin emplear estas restricciones generales. Para comprender el motivo, considérese la relación Trabaja_en obtenida mediante la traducción del diagrama ER en relaciones. Contiene los campos *dni* e *idd*, que son claves externas que hacen referencia a Empleados y a Departamentos. Para asegurar la participación total de los Departamentos en Trabaja_en hay que garantizar que todos los valores de *idd* de Departamentos aparezcan en alguna tupla de Trabaja_en. Se puede intentar garantizar esta condición declarando que *idd* en Departamentos sea una clave externa que haga referencia a Trabaja_en, pero no se trata de una restricción de clave externa válida, ya que *idd* no es clave candidata de Trabaja_en.

Para asegurar la participación total de Departamentos en Trabaja_en mediante SQL, se necesita un aserto. Hay que garantizar que todos los valores de *idd* de Departamentos aparezcan en alguna tupla de Trabaja_en; además, esa tupla de Trabaja_en también debe tener valores no *null* en los campos que sean claves externas que hagan referencia a otros conjuntos de entidades implicados en la relación (en este ejemplo, el campo *dni*). La segunda parte de esta restricción se puede garantizar imponiendo el requisito más estricto de que *dni* en Trabaja_en no pueda contener valores *null*. (Garantizar que la participación de Empleados en Trabaja_en sea total es simétrico.)

Otra restricción que necesita de asertos para expresarse en SQL es el requisito de que cada entidad Empleados (en el contexto del conjunto de relaciones Dirige) debe dirigir, como mínimo, un departamento.

De hecho, el conjunto de relaciones Dirige ejemplifica la mayor parte de las restricciones de participación que se pueden capturar mediante restricciones de clave y de clave externa. Dirige es un conjunto de relaciones binarias en el que exactamente uno de los conjuntos de entidades (Departamentos) tiene una restricción de clave, y la restricción de participación total se expresa en ese conjunto de entidades.

También se pueden capturar restricciones de participación mediante las restricciones de clave y de clave externa en otra situación especial: un conjunto de relaciones en el que todos los conjuntos de entidades participantes tengan restricciones de clave y participación total.

El mejor enfoque de la traducción en este caso es asignar todas las entidades, así como el conjunto de relaciones, a una sola tabla; los detalles son evidentes.

3.5.5 Traducción de los conjuntos de entidades débiles

Los conjuntos de entidades débiles siempre participan en relaciones binarias de uno a varios y tienen una restricción de clave y participación total. El segundo enfoque de la traducción estudiado en el Apartado 3.5.3 es ideal en este caso, pero hay que tener en cuenta que la entidad débil sólo tiene una clave parcial. Además, cuando se elimina una entidad propietaria, se desea que se eliminen todas las entidades débiles de su propiedad.

Considérese el conjunto de entidades débiles Beneficiarios mostrado en la Figura 3.14, con la clave parcial *nombrep*. Cada entidad de Beneficiarios sólo se puede identificar de manera unívoca si se toma la clave de la entidad *propietaria* de Empleados y el *nombrep* de la entidad de Beneficiarios, y hay que eliminar la entidad de Beneficiarios si se elimina la entidad propietaria de Empleados.

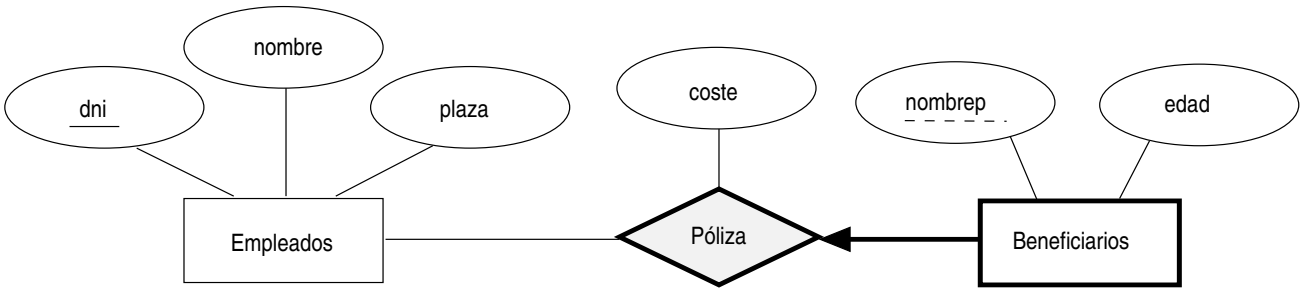


Figura 3.14 El conjunto de entidades débiles Beneficiarios

Se puede capturar la semántica deseada con la siguiente definición de la relación Póliza_dep:

```
CREATE TABLE Póliza_dep (  nombrep  CHAR(20),
                           edad     INTEGER,
                           coste    REAL,
                           dni      CHAR(11),
                           PRIMARY KEY (nombrep, dni),
                           FOREIGN KEY (dni) REFERENCES Empleados
                                   ON DELETE CASCADE )
```

Obsérvese que la clave principal es ⟨*nombrep, dni*⟩, ya que Beneficiarios es una entidad débil. Esta restricción es una modificación con respecto a la traducción tratada en el Apartado 3.5.3. Hay que garantizar que todas las entidades de Departamentos estén asociadas con una entidad de Empleados (la propietaria), al igual que para la restricción de participación total de Beneficiarios. Es decir, *dni* no puede ser *null*. Esto está garantizado porque *dni* forma parte de la clave principal. La opción `CASCADE` asegura que la información sobre la póliza y los beneficiarios se elimine si se elimina la tupla correspondiente de Empleados.

3.5.6 Traducción de las jerarquías de clase

Se presentarán los dos enfoques básicos de la traducción de las jerarquías ES aplicándolos al diagrama ER mostrado en la Figura 3.15:

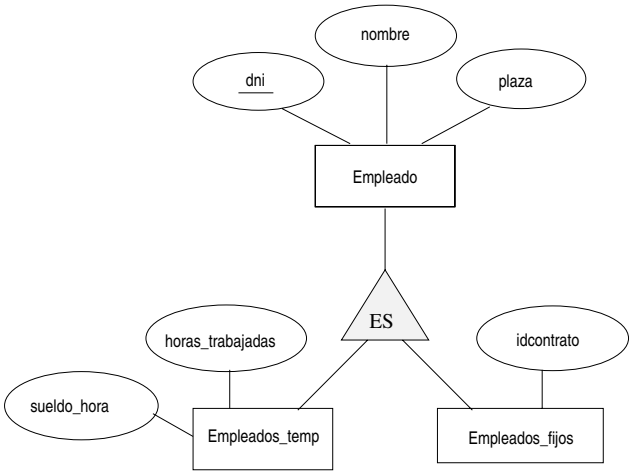


Figura 3.15 Jerarquía de clases

1. Se puede asignar cada uno de los conjuntos de entidades Empleados, Empleados_temp y Empleados_fijos a una relación diferente. La relación Empleados se crea como en el Apartado 2.2. Aquí se estudiará Empleados_temp; Empleados_fijos se maneja de manera parecida. La relación para Empleados_temporales incluye los atributos *sueldo_hora* y *horas_trabajadas* de Empleados_temporales. También contiene los atributos de la clave de la superclase (*dni*, en este ejemplo), que sirve de clave principal para Empleados_temp, así como una clave externa que hace referencia a la superclase (Empleados). Para cada entidad Empleados_temp se almacena el valor de los atributos *nombre* y *plaza* en la fila correspondiente de la superclase (Empleados). Obsérvese que, si se elimina la tupla de la superclase, esa eliminación debe transmitirse en cascada hasta Empleados_temp.

2. De manera alternativa se pueden crear dos relaciones, que se corresponden con Empleados_temp y Empleados_fijos. La relación para Empleados_temp incluye todos los atributos de Empledos_temp, así como todos los atributos de Empleados (es decir, *dni, nombre, plaza, sueldo_hora* y *horas_trabajadas*).

El primer enfoque es general y siempre aplicable. Las consultas en las que se desee examinar todos los empleados y que no se preocupen por los atributos específicos de las subclases se tratan fácilmente mediante la relación Empleados. Sin embargo, puede que las consultas en las que se desee examinar los empleados temporales, por ejemplo, exijan que se combine Empleados_temp (o Empleados_fijos, según el caso) con Empleados para recuperar *nombre* y *plaza*.

El segundo enfoque no es aplicable si hay empleados que no son temporales ni fijos, ya que no hay manera de almacenarlos. Además, si un empleado es a la vez una entidad de Empleados_temp y de Empleados_fijos, los valores de *nombre* y de *plaza* se almacenan dos veces.

Esta duplicación puede provocar alguna de las anomalías que pueden verse en el Capítulo 12. Las consultas que necesiten examinar todos los empleados deben examinar ahora dos relaciones. Por otro lado, una consulta que necesite examinar sólo los empleados temporales lo puede hacer ahora examinando sólo una relación. La elección entre estos enfoques depende claramente de la semántica de los datos y de la frecuencia de las operaciones más comunes.

En general, las restricciones de solapamiento y de cobertura sólo pueden expresarse en SQL mediante asertos.

3.5.7 Traducción de los diagramas ER con agregación

Considérese el diagrama ER que puede verse en la Figura 3.16. Los conjuntos de entidades

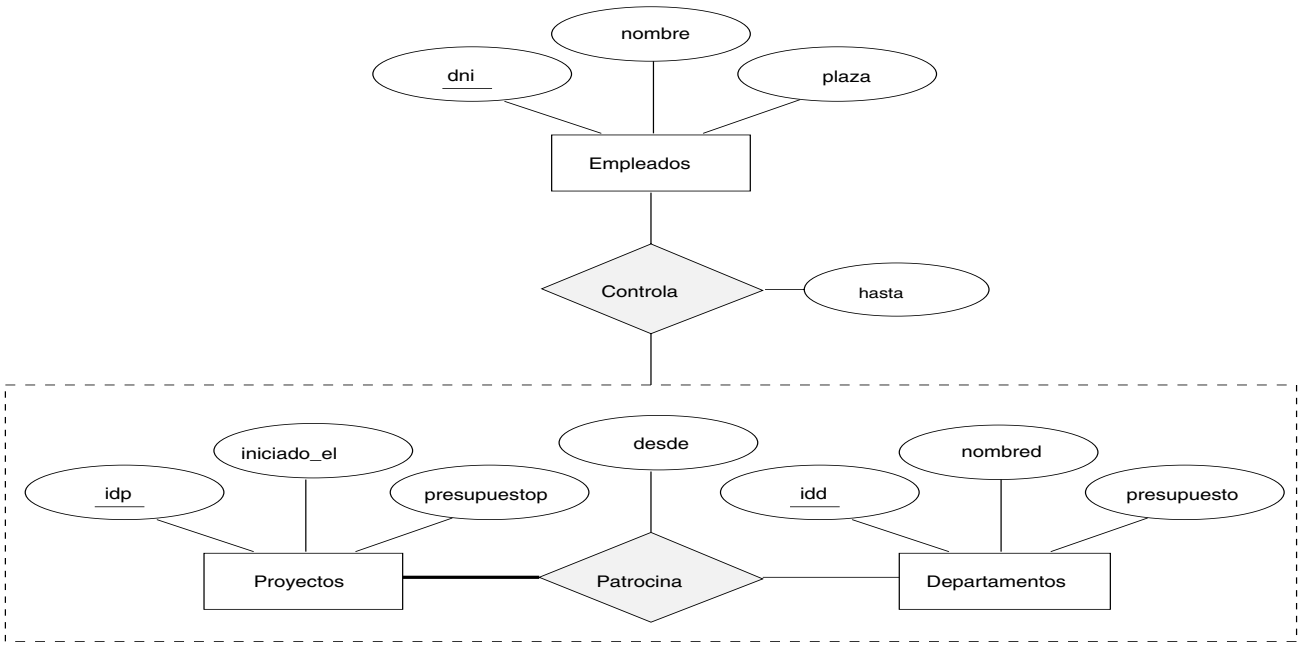


Figura 3.16 Agregación

Empleados, Proyectos y Departamentos y el conjunto de relaciones Patrocina se asignan como se describe en los apartados anteriores. Para el conjunto de relaciones Controla se crea una relación con los atributos siguientes: los de la clave de Empleados (*dni*), los de Patrocina (*idd, idp*) y los atributos descriptivos de Controla (*hasta*). Esta traducción es, esencialmente, la asignación estándar de los conjuntos de relaciones, como se describe en el Apartado 3.5.2.

Hay un caso especial en el que esta traducción se puede refinar, descartando la relación Patrocina. Considérese la relación Patrocina, que tiene los atributos *idp, idd* y *desde*; y, en general, la necesitamos (junto con Controla) por dos razones:

1. Hay que registrar los atributos descriptivos (en este ejemplo, *desde*) de la relación Patrocina.

2. No todos los patrocinios tienen un controlador y, por tanto, puede que algunos pares $\langle idp, idd \rangle$ de la relación Patrocina no aparezcan en la relación Controla.

Sin embargo, si Patrocina no tiene atributos descriptivos y sí una participación total en Controla, se pueden obtener todos los ejemplares posibles de la relación Patrocina a partir de las columnas $\langle idp,\ idd \rangle$ de Controla; por tanto, se puede descartar Patrocina.

3.5.8 Del modelo ER al relacional: más ejemplos

Considérese el diagrama ER que muestra la Figura 3.17. Se pueden emplear las restricciones

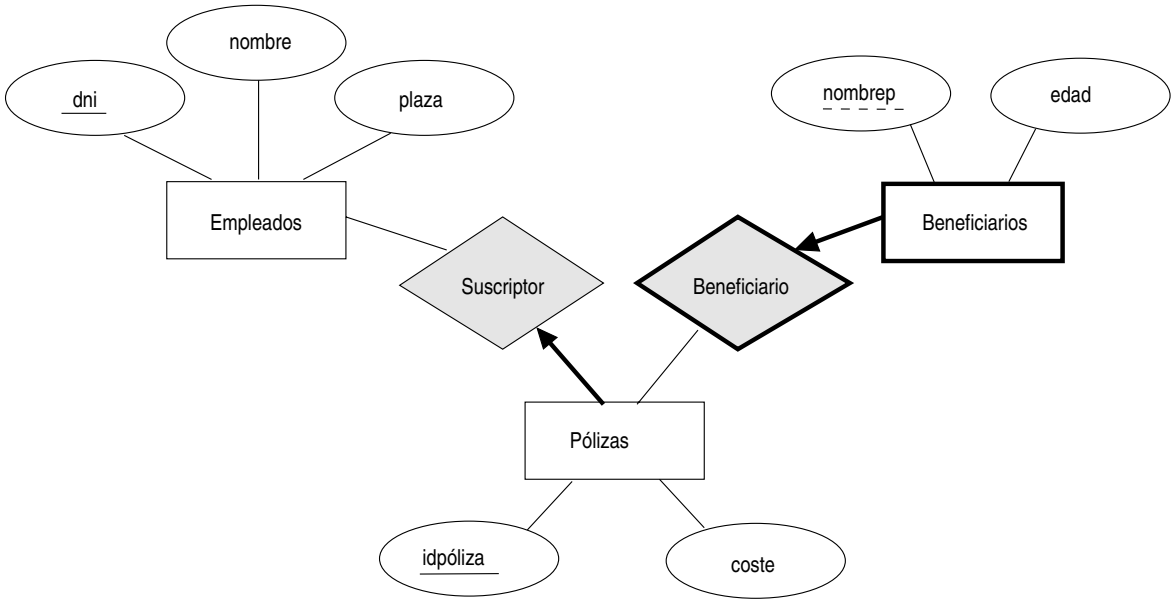


Figura 3.17 Las pólizas de nuevo

de clave para combinar la información de Suscriptor con Pólizas, y la de Beneficiario con la de Beneficiarios y traducirlo todo en el modelo relacional de la manera siguiente:

```
CREATE TABLE Pólizas ( idpóliza  INTEGER,
                       coste     REAL,
                       dni       CHAR(11) NOT NULL,
                       PRIMARY KEY (idpóliza),
                       FOREIGN KEY (dni) REFERENCES Empleados
                               ON DELETE CASCADE )

CREATE TABLE Beneficiarios ( nombrep CHAR(20),
                             edad    INTEGER,
                             idpóliza INTEGER,
                             PRIMARY KEY (nombrep, idpóliza),
                             FOREIGN KEY (idpóliza) REFERENCES Pólizas
                                     ON DELETE CASCADE )
```

Obsérvese cómo la eliminación de un empleado provoca la eliminación de todas las pólizas suscritas por ese empleado y la de todos sus beneficiarios. Además, se exige que cada

beneficiario tenga una póliza que lo cubra —ya que *idpóliza* forma parte de la clave principal de Beneficiarios, hay una restricción `NOT NULL` implícita—. Este modelo refleja con precisión las restricciones de participación del diagrama ER y las acciones deseadas cuando se elimina alguna entidad empleado.

En general puede haber una cadena de relaciones identificadoras para los conjuntos de entidades débiles. Por ejemplo, se ha supuesto que *idpóliza* identifica de manera unívoca a cada póliza. Supóngase que *idpóliza* sólo distingue las pólizas suscritas por un empleado dado; es decir, *idpóliza* sólo es una clave parcial y Pólizas se debe modelar como conjunto de entidades débiles. Esta nueva suposición sobre *idpóliza* no provoca grandes cambios en el estudio anterior. De hecho, las únicas modificaciones son que la clave primaria de Pólizas pasa a ser $\langle$*idpóliza, dni*$\rangle$, y, en consecuencia, la definición de Beneficiarios se modifica —se añade el campo denominado *dni* y pasa a formar parte tanto de la clave principal de Beneficiarios como de la clave externa que hace referencia a Pólizas—:

```
CREATE TABLE Beneficiarios ( nombrep CHAR(20),
                             dni     CHAR(11),
                             edad    INTEGER,
                             idpóliza INTEGER NOT NULL,
                             PRIMARY KEY (nombrep, idpóliza, dni),
                             FOREIGN KEY (idpóliza, dni) REFERENCES Pólizas
                                     ON DELETE CASCADE )
```

3.6 INTRODUCCIÓN A LAS VISTAS

Una **vista** es una tabla cuyas filas no se almacenan en la base de datos de manera explícita, sino que se calculan cuando hace falta a partir de la **definición de la vista**. Considérense las relaciones Alumnos y Matriculado. Supóngase que solemos tener interés en averiguar el nombre y el identificador de alumno de los alumnos que han obtenido un notable en alguna asignatura, junto con el identificador de esa asignatura. Se puede definir una vista con esa finalidad. Empleando la notación de SQL:

```
CREATE VIEW Alumnos-Notable (nombre, ide, asignatura)
       AS SELECT A.nombre, A.ide, M.ida
          FROM   Alumnos A, Matriculado M
          WHERE  A.ide = M.idalum AND M.nota = 'NT'
```

La vista Alumnos-Notable tiene tres campos denominados *nombre*, *ide* y *asignatura* con los mismos dominios que los campos *nombrea* e *ide* de *Alumnos* e *ida* de Matriculado. (Si se omiten los argumentos opcionales *nombre*, *ide*, y *asignatura* de la instrucción `CREATE VIEW`, se heredarán los nombres de columna *nombre*, *ide* e *ida*.)

Esta vista se puede emplear igual que si fuera una **tabla base**, o tabla almacenada explícitamente, para la definición de consultas o de vistas nuevas. Dados los ejemplares de Matriculado y de Alumnos que pueden verse en la Figura 3.4, Alumnos-Notable contendrá las tuplas que aparecen en la Figura 3.18. Conceptualmente, siempre que se emplee Alumnos-Notable, se evalúa en primer lugar la definición de la vista para obtener el ejemplar correspondiente de Alumnos-Notable, luego se evalúa el resto de la consulta tratando a Alumnos-Notable igual

que a cualquier otra relación a la que se haga referencia en la consulta. (La manera en que las consultas sobre las vistas se evalúan en la práctica se trata en el Capítulo 16.)

nombre	*ide*	*asignatura*
Jiménez	53666	Historia105
García	53832	Reggae203

Figura 3.18 Un ejemplar de la vista Alumnos-Notable

3.6.1 Vistas, independencia de datos y seguridad

Considérense los niveles de abstracción que se examinaron en el Apartado 1.5.2. El esquema *físico* de las bases de datos relacionales describe el modo en que se almacenan las relaciones del esquema conceptual en término de la organización de los archivos y de los índices empleados. El esquema *conceptual* es el conjunto de esquemas de las relaciones almacenadas en la base de datos. Aunque algunas relaciones del esquema conceptual también se pueden exponer a las aplicaciones, es decir, formar parte del esquema *externo* de la base de datos, se pueden definir más relaciones en el esquema *externo* mediante el mecanismo de vistas. El mecanismo de vistas proporciona, por tanto, el soporte para la *independencia lógica de datos* en el modelo relacional. Es decir, se puede emplear para definir relaciones en el esquema externo que oculten a las aplicaciones las modificaciones del esquema conceptual de la base de datos. Por ejemplo, si se modifica el esquema de una relación almacenada, se puede definir una vista con el esquema antiguo y las aplicaciones que esperen ver el esquema antiguo podrán utilizar esa vista.

Las vistas también resultan valiosas en el contexto de la *seguridad*: se pueden definir vistas que concedan acceso a cada grupo de usuarios solamente a la información que se les permite ver. Por ejemplo, se puede definir una vista que permita a los alumnos ver el nombre y la edad de los otros alumnos, pero no su nota, y que permita a todos los alumnos tener acceso a esta vista pero no a la tabla Alumnos subyacente (véase el Capítulo 14).

3.6.2 Actualizaciones de las vistas

La motivación subyacente al mecanismo de vistas es la personalización del modo en que los usuarios ven los datos. Los usuarios no deben tener que preocuparse por la distinción entre vistas y tablas base. Este objetivo se consigue realmente en el caso de las consultas a las vistas; las vistas se pueden utilizar igual que cualquier otra relación para la definición de consultas. No obstante, es natural desear especificar también las actualizaciones de las vistas. En este caso, por desgracia, la distinción entre vistas y tablas base se debe tener presente.

La norma SQL-92 sólo permite que se especifiquen las actualizaciones en las vistas que se definan sobre una única tabla base y empleando solamente la selección y la proyección, sin

utilizar operaciones de agregación[4]. Estas vistas se denominan **vistas actualizables**. Esta definición está simplificada en exceso, pero captura el espíritu de las restricciones. Siempre se puede implementar una actualización de estas vistas restringidas mediante la actualización de la tabla base subyacente de una manera no ambigua. Considérese la vista siguiente:

```
CREATE VIEW BuenosAlumnos (ide, nota)
       AS SELECT A.ide, A.nota
          FROM   Alumnos A
          WHERE  A.nota > 6.0
```

Se puede implementar una orden que modifique la nota de una fila de BuenosAlumnos mediante la modificación de la fila correspondiente de Alumnos. Se puede eliminar una fila de BuenosAlumnos mediante la eliminación de la fila correspondiente de Alumnos. (Por lo general, si la vista no incluye ninguna clave para la tabla subyacente, varias filas de la tabla podrían "corresponder" a una sola fila de la vista. Éste sería el caso, por ejemplo, si se empleara $A.nombre$ en lugar de $A.ide$ en la definición de BuenosAlumnos. Cualquier orden que afecte a una fila de la vista afectará a todas las filas correspondientes de la tabla subyacente.)

Se puede insertar una fila de BuenosAlumnos mediante la inserción de una fila en Alumnos, empleando valores *null* en las columnas de Alumnos que no aparezcan en BuenosAlumnos (por ejemplo, *nombre* y *usuario*). Obsérvese que no se permite que las columnas de la clave primaria contengan valores *null*. Por tanto, si se intenta insertar filas mediante alguna vista que no contenga la clave primaria de la tabla subyacente, se rechazarán esas inserciones. Por ejemplo, si BuenosAlumnos contuviera *nombre* pero no *ide*, no se podrían insertar filas en Alumnos mediante inserciones en BuenosAlumnos.

Una observación importante es que una instrucción `INSERT` o `UPDATE` puede modificar la tabla base subyacente de modo que la fila resultante (es decir, insertada o modificada) no se halle en la vista. Por ejemplo, si se intenta insertar la fila $\langle 51234,\ 5{,}6 \rangle$ en la vista, esta fila se puede (rellenar con valores *null* en los demás campos de Alumnos y luego) añadir a la tabla Alumnos subyacente, pero no aparecerá en la vista BuenosAlumnos, ya que no satisface la condición de la vista $nota > 6,0$. La acción predeterminada de SQL es permitir esta inserción, pero se puede impedir añadiendo la cláusula `WITH CHECK OPTION` a la definición de la vista. En este caso, sólo son posibles las inserciones de las filas que aparecen realmente en la vista.

Se advierte al lector de que, cuando se define una vista en términos de otra, la interacción entre las definiciones de ambas vistas con respecto a las actualizaciones y a la cláusula `CHECK OPTION` puede ser compleja; no se entrará aquí en más detalles.

Necesidad de restringir las actualizaciones de las vistas

Aunque las reglas de SQL sobre las vistas actualizables son más estrictas de lo necesario, hay algunos problemas fundamentales con las actualizaciones especificadas para las vistas que son una buena razón para limitar las clases de vistas que se pueden actualizar. Considérese la relación Alumnos y una relación nueva denominada Clubs:

[4]También existe la restricción de que no se puede emplear el operador `DISTINCT` en la definición de vistas actualizables. De manera predeterminada, SQL no elimina las copias duplicadas de las filas del resultado de las consultas; el operador `DISTINCT` exige la eliminación de los duplicados. Este aspecto se analizará con más detalle en el Capítulo 5.

Vistas actualizables en SQL:1999 La nueva norma de SQL ha ampliado la clase de definiciones de vistas que son actualizables, teniendo en cuenta las restricciones de clave principal. A diferencia de SQL-92, según la nueva definición se pueden actualizar las definiciones de vistas que contengan más de una tabla en la cláusula `FROM`. De manera intuitiva, se pueden actualizar los campos de una vista que tan sólo se obtengan de una de las tablas subyacentes, y la clave principal de esa tabla se incluirá en los campos de esa vista.

SQL:1999 distingue entre las vistas cuyas filas se pueden modificar (*vistas actualizables*) y las vistas en las que se pueden insertar filas nuevas (**vistas insertables**): las vistas definidas mediante las estructuras de SQL `UNION`, `INTERSECT` y `EXCEPT` (que se tratarán en el Capítulo 5) no se pueden insertar, aunque sean actualizables. De manera intuitiva, la actualizabilidad garantiza que cada tupla actualizada de la vista se pueda rastrear hasta exactamente una tupla de las tablas empleadas para crear esa vista. Puede que la propiedad de actualizabilidad, sin embargo, no permita determinar la tabla en la que insertar la nueva tupla.

nombrec	*añoi*	*nombres*
Vela	1996	Díaz
Senderismo	1997	Sánchez
Remo	1998	Sánchez

Figura 3.19 El ejemplar C de Clubs

ide	*nombre*	*usuario*	*edad*	*nota*
50000	Díaz	diaz@inf	19	6,6
53666	Jiménez	jimenez@inf	18	6,8
53688	Sánchez	sanchez@ii	18	6,4
53650	Sánchez	sanchez@mat	19	7,6

Figura 3.20 El ejemplar $A3$ de Alumnos

Clubs(*nombrec:* `string`, *añoi:* `date`, *nombres:* `string`)

nombres ha sido socio del club *nombrec* desde la fecha *añoi*[5]. Supóngase que a menudo estamos interesados en averiguar el nombre y el usuario de los alumnos con nota superior a 6 que pertenecen, como mínimo a un club, además del nombre del club y de la fecha en que ingresaron en él. Se puede definir una vista con esta finalidad:

```
CREATE VIEW AlumnosActivos (nombre, usuario, club, desde)
       AS SELECT A.nombre, A.usuario, C.nombrec, C.añoi
          FROM   Alumnos A, Clubs C
          WHERE  A.nombre = C.nombres AND A.nota > 6
```

Considérense los ejemplares de Alumnos que pueden verse en las Figuras 3.19 y 3.20. Cuando se evalúa empleando los ejemplares C y $A3$, AlumnosActivos contiene las filas que pueden verse en la Figura 3.21.

Supóngase ahora que se desea eliminar la fila ⟨*Sánchez, sanchez@ii, Senderismo, 1997*⟩ de AlumnosActivos. ¿Cómo hacerlo? Las filas de AlumnosActivos no se almacenan de manera

[5]Hay que destacar que Clubs tiene un esquema mal diseñado (escogido en aras del estudio de las actualizaciones de las vistas), ya que identifica a los alumnos por su nombre, que no es clave candidata de Alumnos.

nombre	*usuario*	*club*	*desde*
Díaz	diaz@inf	Vela	1996
Sánchez	sanchez@ii	Senderismo	1997
Sánchez	sanchez@ii	Remo	1998
Sánchez	sanchez@mat	Senderismo	1997
Sánchez	sanchez@mat	Remo	1998

Figura 3.21 Ejemplar de AlumnosActivos

explícita, sino que se calculan según hacen falta a partir de las tablas Alumnos y Clubs mediante la definición de la vista. Por tanto, hay que modificar Alumnos o Clubs (o ambas) de manera que la evaluación de la definición de la vista para el ejemplar modificado no genere la fila ⟨*Sánchez, sanchez@ii, Senderismo, 1997*⟩. Esta tarea se puede cumplir de dos maneras: eliminando la fila ⟨*53688, Sánchez, sanchez@ii, 18, 6,4*⟩ de Alumnos o eliminando la fila ⟨*Senderismo, 1997, Sánchez*⟩ de Clubs. Pero ninguna de estas soluciones es satisfactoria. La eliminación de la fila de Alumnos tiene el efecto de eliminar también la fila ⟨*Sánchez, sanchezh@ii, Remo, 1998*⟩ de la vista AlumnosActivos. La eliminación de la fila Clubs tiene el efecto de eliminar también la fila ⟨*Sánchez, sanchez@mat, Senderismo, 1997*⟩ de la vista AlumnosActivos. Ninguno de estos efectos secundarios es deseable. De hecho, la única solución razonable es *impedir* esas actualizaciones de las vistas.

Las vistas que implican a más de una tabla base, en principio, se pueden actualizar con seguridad. La vista Alumnos-Notable que se introdujo al principio de este apartado es un ejemplo de ese tipo de vistas. Considérese el ejemplar de Alumnos-Notable que aparece en la Figura 3.18 (por supuesto, con los ejemplares correspondientes de Alumnos y de Matriculado, de la Figura 3.4). Para insertar una tupla, por ejemplo, ⟨*Díaz, 50000, Reggae203*⟩ en Alumnos-Notable basta con insertar la tupla ⟨*Reggae203, NT, 50000*⟩ en Matriculado, puesto que ya existe una tupla para *ide* 50000 en Alumnos. Para insertar ⟨*Juan, 55000, Reggae203*⟩, por otro lado, hay que insertar ⟨*Reggae203, NT, 55000*⟩ en Matriculado y también ⟨*55000, Juan,* null*,* null*,* null⟩ en Alumnos. Obsérvese cómo se emplean los valores *null* en los campos de la tupla insertada cuyo valor no está disponible. Afortunadamente, el esquema de la vista contiene los campos de la clave primaria de las dos tablas base subyacentes; en caso contrario, no se podrían admitir las inserciones en esta vista. Para eliminar una tupla de la vista Alumnos-Notable basta con eliminar la tupla correspondiente de Matriculado.

Aunque este ejemplo ilustre que las reglas de SQL para las vistas actualizables resultan innecesariamente restrictivas, también pone de manifiesto la complejidad del manejo de las actualizaciones de las vistas en un caso general. Por motivos prácticos, la norma de SQL ha decidido permitir únicamente las actualizaciones para una clase muy restringida de vistas.

3.7 ELIMINACIÓN Y MODIFICACIÓN DE TABLAS Y VISTAS

Si se decide que ya no hace falta una tabla base y se desea eliminarla (por ejemplo, borrar todas sus filas *y* quitar la información de definición de la tabla), se puede utilizar la orden

DROP TABLE. Por ejemplo, DROP TABLE Alumnos RESTRICT elimina la tabla Alumnos a menos que alguna vista o restricción de integridad haga referencia a Alumnos; en ese caso, la orden fallará. Si se sustituye la palabra clave RESTRICT por CASCADE, se eliminan (de manera recursiva) Alumnos y cualquier vista o restricción de integridad que haga referencia a ella; siempre hay que especificar una de estas dos palabras clave. Se puede eliminar una vista empleando la orden DROP VIEW, que es igual que DROP TABLE.

ALTER TABLE modifica la estructura de una tabla ya existente. Para añadir a Alumnos la columna denominada *teléfono*, por ejemplo, se usaría la orden siguiente:

```
ALTER TABLE Alumnos
      ADD COLUMN teléfono CHAR(10)
```

Se modifica la definición de Alumnos para añadir esta columna, que en todas las filas ya existentes se rellena con valores *null*. ALTER TABLE también se puede emplear para eliminar columnas y añadir o eliminar restricciones de integridad en las tablas; estos aspectos de la orden no se estudiarán más allá de destacar que la eliminación de columnas se trata de manera muy parecida a la de tablas o vistas.

3.8 ESTUDIO DE UN CASO: LA TIENDA EN INTERNET

La siguiente etapa del diseño de este ejemplo, que viene del Apartado 2.8, es el diseño lógico de la base de datos. Empleando el enfoque habitual tratado en este capítulo, TiposBD obtienen las siguientes tablas en el modelo relacional a partir del diagrama ER de la Figura 2.20:

```
CREATE TABLE Libros ( isbn             CHAR(10),
                      título           CHAR(80),
                      autor            CHAR(80),
                      stock            INTEGER,
                      precio           REAL,
                      año_publicación  INTEGER,
                      PRIMARY KEY (isbn))

CREATE TABLE Pedidos ( isbn          CHAR(10),
                       idc           INTEGER,
                       tarjeta       CHAR(16),
                       cantidad      INTEGER,
                       fecha_pedido  DATE,
                       fecha_envío   DATE,
                       PRIMARY KEY (isbn,idc),
                       FOREIGN KEY (isbn) REFERENCES Libros,
                       FOREIGN KEY (idc) REFERENCES Clientes )

CREATE TABLE Clientes ( idc        INTEGER,
                        nombrec    CHAR(80),
                        dirección  CHAR(200),
                        PRIMARY KEY (idc))
```

El jefe del equipo de diseño, que sigue dándole vueltas al hecho de que la revisión expuso un fallo del diseño, tiene ahora una inspiración. La tabla Pedidos contiene el campo *fecha_pedido* y la clave de la tabla sólo contiene los campos *isbn* e *idc*. Debido a esto, los clientes no pueden pedir el mismo libro en días diferentes, una restricción no deseada. ¿Por qué no añadir el atributo *fecha_pedido* a la clave de la tabla Pedidos? Eso eliminaría la restricción no deseada:

```
CREATE TABLE Pedidos (    isbn          CHAR(10),
                          ...
                          PRIMARY KEY (isbn,idc,fecha_envío),
                          ...)
```

El revisor, Tipo 2, no está contento del todo con esta solución, que denomina "parche". Señala que ningún diagrama ER natural refleja este diseño y subraya la importancia del diagrama ER como documento de diseño. Tipo 1 arguye que, aunque Tipo 2 tiene algo de razón, es importante presentar a B&N un diseño preliminar y obtener una respuesta; todos están de acuerdo en esto, y vuelven a B&N.

El propietario de B&N presenta ahora algunos requisitos adicionales que no mencionó durante las discusiones iniciales: "Los clientes deben poder comprar varios libros diferentes en cada pedido. Por ejemplo, si un cliente desea comprar tres copias de 'El profesor de español' y dos copias de 'El carácter de las leyes físicas', debe poder formular un solo pedido por los dos títulos."

El jefe del equipo de diseño, Tipo 1, pregunta el modo en que eso afecta a la política de envíos. ¿Sigue deseando B&N enviar juntos todos los libros del mismo pedido? El propietario de B&N explica su política de envíos: "En cuanto se disponga de copias suficientes del libro encargado, se envía, aunque un pedido contenga varios libros. Por tanto, puede ocurrir que las tres copias de 'El profesor de español' se envíen hoy porque haya cinco copias en stock pero que 'El carácter de las leyes físicas' se envíe mañana, porque sólo se disponga actualmente de una copia en stock y mañana llegue otra copia. Además, los clientes pueden formular más de un pedido al día, y desean poder identificar los pedidos que han formulado."

El equipo de TiposBD vuelve a meditar sobre esto e identifica dos requisitos nuevos. En primer lugar, debe ser posible encargar varios libros diferentes en un mismo pedido y, en segundo lugar, cada cliente debe poder distinguir entre los diferentes pedidos que haya formulado en un mismo día. Para adecuarse a estos requisitos, introducen en la tabla Pedidos un atributo nuevo denominado *númpedido*, que identifica de manera unívoca cada pedido y, por tanto, al cliente que lo ha formulado. No obstante, dado que se pueden comprar varios libros en un mismo pedido, tanto *númpedido* como *isbn* son necesarios para determinar *cant* y *fecha_envío* en la tabla Pedidos.

A los pedidos se les asignan secuencialmente números de pedido, y los que se formulan más tarde tienen números de pedido mayores. Si el mismo cliente formula varios pedidos el mismo día, esos pedidos tendrán números de pedido diferentes y, por tanto, podrán distinguirse. La instrucción del LDD de SQL para crear la tabla Pedidos modificada es la siguiente:

```
CREATE TABLE Pedidos ( númpedido   INTEGER,
                       isbn        CHAR(10),
                       idc         INTEGER,
                       tarjeta     CHAR(16),
```

```
cantidad      INTEGER,
fecha_pedido  DATE,
fecha_envío   DATE,
PRIMARY KEY (númpedido, isbn),
FOREIGN KEY (isbn) REFERENCES Libros
FOREIGN KEY (idc) REFERENCES Clientes )
```

El propietario de B&N está bastante contento con este diseño para Pedidos, pero se ha dado cuenta de otra cosa. (Los de TiposBD no están sorprendidos; los clientes siempre llegan con requisitos nuevos a medida que avanza el diseño.) Aunque desea que todos sus empleados puedan examinar los detalles de los pedidos, de modo que puedan responder a las consultas de los clientes, quiere que la información relativa a las tarjetas de crédito de los clientes permanezca segura. Para abordar este requisito, TiposBD crea la vista siguiente:

```
CREATE VIEW InfoPedido (isbn, idc, cantidad, fecha_pedido, fecha_envío)
       AS SELECT P.idc, P.cantidad, P.fecha_pedido, P.fecha_envío
          FROM   Pedidos P
```

El plan es permitir que los empleados vean esta tabla, pero no la tabla Pedidos; la última queda restringida al departamento de Contabilidad de B&N. Se verá la manera de conseguirlo en el Apartado 14.7.

3.9 PREGUNTAS DE REPASO

Las respuestas a las preguntas de repaso se pueden hallar en los apartados señalados.

- ¿Qué es una relación? Explíquese la diferencia entre esquema de relación y ejemplar de relación. Defínanse los términos *aridad* y *grado* de una relación. ¿Qué son las restricciones de dominio? **(Apartado 3.1)**
- ¿Qué estructura de SQL permite la definición de relaciones? ¿Qué estructuras permiten la modificación de los ejemplares de las relaciones? **(Apartado 3.1.1)**
- ¿Qué son las *restricciones de integridad*? Defínanse los términos *restricción de clave principal* y *restricción de clave externa*. ¿Cómo se expresan estas restricciones en SQL? ¿Qué otros tipos de restricciones se pueden expresar en SQL? **(Apartado 3.2)**
- ¿Qué hace el SGBD cuando se violan las restricciones? ¿Qué es la *integridad referencial*? ¿Qué opciones da SQL a los programadores de aplicaciones para que respondan a las violaciones de la integridad referencial? **(Apartado 3.3)**
- ¿Cuándo hace cumplir el SGBD las restricciones de integridad? ¿Cómo pueden los programadores de aplicaciones controlar el momento en que se comprueban las violaciones de las restricciones durante la ejecución de las transacciones? **(Apartado 3.3.1)**
- ¿Qué es una *consulta a una base de datos relacional*? **(Apartado 3.4)**

- ¿Cómo se pueden traducir los diagramas ER en instrucciones de SQL para crear tablas? ¿Cómo se traducen los conjuntos de entidades en relaciones? ¿Cómo se traducen los conjuntos de relaciones? ¿Cómo se manejan las restricciones del modelo ER, los conjuntos de entidades débiles, las jerarquías de clases y la agregación? **(Apartado 3.5)**
- ¿Qué es una *vista*? ¿Cómo soportan las vistas la independencia lógica de los datos? ¿Cómo se emplean las vistas para mejorar la seguridad? ¿Cómo se evalúan las consultas sobre las vistas? ¿Por qué restringe SQL la clase de vistas que se pueden actualizar? **(Apartado 3.6)**
- ¿Cuáles son las estructuras de SQL que modifican la estructura de las tablas y eliminan tablas y vistas? Discútase lo que ocurre cuando se elimina una vista. **(Apartado 3.7)**

EJERCICIOS

Ejercicio 3.1 Defínanse los términos siguientes: *esquema de una relación, esquema de una base de datos relacional, dominio, atributo, dominio de un atributo, ejemplar de una relación, cardinalidad de una relación* y *grado de una relación*.

Ejercicio 3.2 ¿Cuántas tuplas distintas hay en un ejemplar de una relación con cardinalidad 22?

Ejercicio 3.3 ¿Ofrece el modelo relacional, tal y como lo ve un escritor de consultas de SQL, independencia física y lógica de datos? Explíquese.

Ejercicio 3.4 ¿Cuál es la diferencia entre una clave candidata y la clave principal de una relación dada? ¿Qué es una superclave?

Ejercicio 3.5 Considérese el ejemplar de la relación Alumnos que puede verse en la Figura 3.1.

1. Dese un ejemplo de atributo (o de conjunto de atributos) que se pueda deducir que *no* es una clave candidata, basándose en que el ejemplar sea legal.
2. ¿Hay algún ejemplo de atributo (o de conjunto de atributos) que se pueda deducir que *es* una clave candidata, basándose en que el ejemplar sea legal?

Ejercicio 3.6 ¿Qué es una restricción de clave externa? ¿Por qué son importantes esas restricciones? ¿Qué es la integridad referencial?

Ejercicio 3.7 Considérense las relaciones Alumnos, Profesores, Asignaturas, Aulas, Matriculado, Imparte e Impartida_en definidas en el Apartado 1.5.2.

1. Cítense todas las restricciones de clave externa existentes en estas relaciones.
2. Dese un ejemplo de una restricción (plausible) que implique a una o a varias de estas relaciones y que no sea ni de clave principal ni de clave externa.

Ejercicio 3.8 Contéstese brevemente a cada una de estas preguntas. Están basadas en el siguiente esquema relacional:

Emp(*ide:* `integer`, *nombree:* `string`, *edad:* `integer`, *sueldo:* `real`)
Trabaja(*ide:* `integer`, *idd:* `integer`, *Parcial:* `integer`)
Dep(*idd:* `integer`, *nombred:* `string`, *presupuesto:* `real`, *idencargado:* `integer`)

1. Dese un ejemplo de restricción de clave externa que implique a la relación Dep. ¿Cuáles son las opciones para hacer que se cumpla esta restricción cuando un usuario intenta eliminar alguna tupla de Dep?

2. Escríbanse las instrucciones de SQL necesarias para crear las relaciones anteriores, incluyendo las versiones adecuadas de todas las restricciones de clave principal y de clave externa.
3. Defínase la relación Dep en SQL de modo que se garantice que todos los departamentos tengan un encargado.
4. Escríbase una instrucción de SQL para añadir a Pedro Pérez como empleado con $ide = 101$, $edad = 32$ y $sueldo = 15000$.
5. Escríbase una instrucción de SQL para incrementar en un 10 por 100 el sueldo de cada empleado.
6. Escríbase una instrucción de SQL para eliminar el departamento de juguetes. Dadas las restricciones de integridad referencial escogidas para este esquema, explíquese lo que ocurre al ejecutar esta instrucción.

Ejercicio 3.9 Considérese la consulta de SQL cuya respuesta se muestra en la Figura 3.6.

1. Modifíquese esta consulta de modo que sólo se incluya en la respuesta la columna *usuario*.
2. Si se añade la cláusula `WHERE` $A.nota >= 4$ a la consulta original, ¿cuál es el conjunto de tuplas de la respuesta?

Ejercicio 3.10 Explíquese el motivo de que al añadir las restricciones `NOT NULL` a la definición SQL de la relación Dirige (del Apartado 3.5.3) no haga que se cumpla la restricción de que cada departamento debe tener un encargado. ¿Qué se consigue, si es que se consigue algo, exigiendo que el campo *dni* de Dirige no sea *null*?

Ejercicio 3.11 Supóngase que se tiene una relación ternaria R entre los conjuntos de entidades A, B y C tal que A tenga una restricción de clave y participación total y B tenga una restricción de clave; éstas son las únicas restricciones. A tiene los atributos $a1$ y $a2$, donde $a1$ es la clave; B y C son parecidas. R no tiene atributos descriptivos. Escríbanse instrucciones de SQL que creen tablas que se correspondan con esta información de modo que capturen tantas restricciones como sea posible. Si no se puede capturar alguna restricción, explíquese el motivo.

Ejercicio 3.12 Considérese la situación del Ejercicio 2.2, en la que se ha diseñado un diagrama ER para la base de datos de una universidad. Escríbanse las instrucciones de SQL para crear las relaciones correspondientes y captúrense tantas restricciones como sea posible. Si no se puede capturar alguna restricción, explíquese el motivo.

Ejercicio 3.13 Considérese la base de datos para la universidad del Ejercicio 2.3 y el diagrama ER que se diseñó. Escríbanse las instrucciones de SQL para crear las relaciones correspondientes y captúrense tantas restricciones como sea posible. Si no se puede capturar alguna restricción, explíquese el motivo.

Ejercicio 3.14 Considérese la situación del Ejercicio 2.4, en la que se diseñó el diagrama ER para la base de datos de una empresa. Escríbanse instrucciones de SQL para crear las relaciones correspondientes y captúrense tantas restricciones como sea posible. Si no se puede capturar alguna, explíquese el motivo.

Ejercicio 3.15 Considérese la base de datos Sinpueblo del Ejercicio 2.5. Se ha decidido recomendar que Sinpueblo emplee un sistema relacional de bases de datos para almacenar los datos de la empresa. Muéstrense las instrucciones de SQL para crear las relaciones correspondientes a los conjuntos de entidades y de relaciones del diseño. Identifíquense las restricciones que haya en el diagrama ER que no se puedan capturar en las instrucciones de SQL y explíquese brevemente el motivo de que no se puedan expresar.

Ejercicio 3.16 Tradúzcase todo el diagrama ER del Ejercicio 2.6 a un esquema relacional y muéstrense las instrucciones SQL necesarias para crear las relaciones empleando únicamente restricciones de clave y *null*. Si la traducción no puede capturar ninguna restricción del diagrama ER, explíquese el motivo.

En el Ejercicio 2.6 también se modificó el diagrama ER para incluir la restricción de que las revisiones de cada avión deben realizarlas técnicos expertos en ese modelo. ¿Se pueden modificar las instrucciones SQL que definen las relaciones obtenidas mediante asignación del diagrama ER para que comprueben esta restricción?

Ejercicio 3.17 Considérese el diagrama ER que se diseñó para la cadena de farmacias Recetas-R-X del Ejercicio 2.7. Defínanse las relaciones correspondientes a los conjuntos de entidades y de relaciones de este diseño mediante SQL.

Ejercicio 3.18 Escríbanse instrucciones SQL para crear las relaciones correspondientes al diagrama ER que se diseñó para el Ejercicio 2.8. Si la traducción no puede capturar ninguna restricción del diagrama ER, explíquese el motivo.

Ejercicio 3.19 Respóndanse brevemente las preguntas siguientes basándose en este esquema:

Emp(*ide:* `integer`, *nombree:* `string`, *edad:* `integer`, *sueldo:* `real`)
Trabaja(*ide:* `integer`, *idd:* `integer`, *tiempo_parcial:* `integer`)
Dep(*idd:* `integer`, *presupuesto:* `real`, *idencargado:* `integer`)

1. Supóngase que se tiene la vista EmpVeterano definida de la manera siguiente:

```
CREATE VIEW EmpVeterano (nombrev, edadv, sueldo)
       AS SELECT E.nombree, E.edad, E.sueldo
          FROM   Emp E
          WHERE  E.edad > 50
```

   Explíquese lo que hará el sistema para procesar la consulta siguiente:

```
SELECT V.nombrev
FROM   EmpVeterano V
WHERE  V.sueldo > 100000
```

2. Dese un ejemplo de vista de Emp que se pueda actualizar de manera automática actualizando Emp.
3. Dese un ejemplo de vista de Emp que sería imposible actualizar (automáticamente) y explíquese el motivo de que el ejemplo presente el problema de actualización que tiene.

Ejercicio 3.20 Considérese el esquema siguiente:

Proveedores(*idp:* `integer`, *nombrep:* `string`, *dirección:* `string`)
Repuestos(*idr:* `integer`, *nombrer:* `string`, *color:* `string`)
Catálogo(*idp:* `integer`, *idr:* `integer`, *coste:* `real`)

La relación Catálogo muestra los precios de los repuestos cobrados por los proveedores. Respóndase a las preguntas siguientes:

- Dese un ejemplo de vista actualizable que implique a una sola relación.
- Dese un ejemplo de vista actualizable que implique a dos relaciones.
- Dese un ejemplo de vista insertable que sea actualizable.
- Dese un ejemplo de vista insertable que no sea actualizable.

EJERCICIOS BASADOS EN UN PROYECTO

Ejercicio 3.21 Créense en Minibase las relaciones Alumnos, Profesores, Asignaturas, Aulas, Matriculado, Imparte e Impartida_en.

Ejercicio 3.22 Insértense las tuplas mostradas en las Figuras 3.1 y 3.4 en las relaciones Alumnos y Matriculado. Créense ejemplares razonables de las otras relaciones.

Ejercicio 3.23 ¿Qué restricciones de integridad hace cumplir Minibase?

Ejercicio 3.24 Ejecútense las consultas de SQL presentadas en este capítulo.

NOTAS BIBLIOGRÁFICAS

El modelo relacional lo propuso Codd en un trabajo pionero [119]. Childs [111] y Kuhns [287] presagiaron algunos de estos desarrollos. El libro de Gallaire y Minker [194] contiene varios trabajos sobre el empleo de la lógica en el contexto de las bases de datos relacionales. En [454] se describe un sistema basado en una variación del modelo relacional en el que toda la base de datos se considera abstractamente como una relación única, denominada *relación universal*. Varios autores discuten extensiones del modelo relacional que incorporan valores *null*, que indican un valor desconocido o ausente para un campo; por ejemplo, [214, 259, 374, 460, 481].

Entre los proyectos pioneros están el Sistema R [25, 93] del Laboratorio de investigación de IBM en San José (actualmente el Centro de Investigación Almadén de IBM), Ingres [435] de la Universidad de California en Berkeley, PRTV [446] del Centro Científico de IBM del Reino Unido en Peterlee y QBE [491] del Centro de investigación T. J. Watson de IBM.

Una rica teoría sostiene el campo de las bases de datos relacionales. Entre los textos dedicados a los aspectos teóricos están los de Atzeni y DeAntonellis [29]; Maier [309]; y Abiteboul, Hull, y Vianu [1]. [265] es un excelente artículo recopilatorio.

Las restricciones de integridad de las bases de datos relacionales se han estudiado a fondo. [122] aborda las extensiones semánticas del modelo relacional y la integridad, en especial la integridad referencial. [234] estudia las restricciones de integridad semánticas. [135] contiene trabajos que abordan diversos aspectos de las restricciones de integridad, incluyendo en especial un estudio detallado de la integridad referencial. Una amplia literatura trata del cumplimiento de las restricciones de integridad referencial. [35] compara el coste del cumplimiento de las restricciones de integridad mediante comprobaciones en el momento de compilación, en el momento de ejecución y posteriores a la ejecución. [89] presenta un lenguaje basado en SQL para especificar restricciones de integridad e identifica las condiciones en las que se pueden violar las reglas de integridad especificadas en este lenguaje. [433] estudia la técnica de comprobación de las restricciones de integridad mediante la modificación de consultas. [114] estudia las restricciones de integridad en tiempo real. Otros trabajos sobre la comprobación de las restricciones de integridad en las bases de datos son [53, 71, 84, 322]. [409] considera el enfoque de comprobar la corrección de los programas que tienen acceso a la base de datos en lugar de las comprobaciones en el momento de la ejecución. Téngase en cuenta que esta lista de referencias dista mucho de estar completa; de hecho, no incluye ninguno de los muchos trabajos sobre la comprobación de restricciones de integridad especificadas de manera recursiva. Algunos de los primeros trabajos en este campo tan estudiado se pueden encontrar en [194] y [193].

Para las referencias sobre SQL, véanse las notas bibliográficas del Capítulo 5. Este libro no estudia productos concretos basados en el modelo relacional, pero muchos buenos libros tratan cada uno de los principales sistemas comerciales; por ejemplo, el libro de Chamberlin sobre DB2 [92], el de Date y McGoveran sobre Sybase [139] y el de Koch y Loney sobre Oracle [282].

Varios trabajos consideran el problema de la traducción de las actualizaciones especificadas en vistas en traducciones de la tabla subyacente [39, 140, 269, 293, 473]. [192] es un buen resumen de este tema. Véanse las notas bibliográficas del Capítulo 16 para obtener referencias sobre el trabajo con vistas de consultas y el mantenimiento de las vistas materializadas.

[443] estudia una metodología de diseño basada en el desarrollo de diagramas ER y su posterior traducción al modelo relacional. Markowitz considera la integridad referencial en el contexto de la asignación del modelo ER al relacional y estudia el soporte ofrecido en algunos sistemas comerciales (de esa época) en [318, 319].