

Bases de datos.

Diseño y gestión

Arturo Mora Rioja

A large, solid blue circle is partially visible on the left side of the page, overlapping the light blue background.

Bases de datos. Diseño y gestión

Consulte nuestra página web: www.sintesis.com
En ella encontrará el catálogo completo y comentado



Queda prohibida, salvo excepción prevista en la ley, cualquier forma de reproducción, distribución, comunicación pública y transformación de esta obra sin contar con autorización de los titulares de la propiedad intelectual. La infracción de los derechos mencionados puede ser constitutiva de delito contra la propiedad intelectual (arts. 270 y sigs. Código Penal). El Centro Español de Derechos Reprográficos (www.cedro.org) vela por el respeto de los citados derechos.

Bases de datos. **Diseño y gestión**

Arturo Mora Rioja

© Arturo Mora Rioja

© EDITORIAL SÍNTESIS, S. A.
Vallehermoso, 34. 28015 Madrid
Teléfono 91 593 20 98
<http://www.sintesis.com>

ISBN: 978-84-907755-9-2

Reservados todos los derechos. Está prohibido, bajo las sanciones penales y el resarcimiento civil previstos en las leyes, reproducir, registrar o transmitir esta publicación, íntegra o parcialmente, por cualquier sistema de recuperación y por cualquier medio, sea mecánico, electrónico, magnético, electroóptico, por fotocopia o por cualquier otro, sin la autorización previa por escrito de Editorial Síntesis, S. A.

Índice

INTRODUCCIÓN	9
1. SISTEMAS DE ALMACENAMIENTO DE LA INFORMACIÓN	13
Objetivos	13
Mapa conceptual del capítulo	14
Glosario.....	15
1.1. Introducción	15
1.2. Ficheros.....	16
1.2.1. Tipos de fichero según su estructura de almacenamiento	17
1.2.2. Tipos de soporte de almacenamiento	19
1.3. Bases de datos	20
1.3.1. Definición	20
1.3.2. Tipos de bases de datos	21
1.4. Sistemas gestores de bases de datos.....	23
1.4.1. Componentes del SGBD	23
1.4.2. Funciones del SGBD	24
Resumen.....	25
Ejercicios propuestos.....	25
Actividades de autoevaluación	26
2. DISEÑO DE BASES DE DATOS RELACIONALES. EL MODELO ENTIDAD/RELACIÓN	27
Objetivos	27
Mapa conceptual del capítulo	28
Glosario.....	29
2.1. Modelización conceptual del <i>software</i>	29
2.1.1. Modelización de datos	30
2.1.2. Diccionario de datos	31
2.1.3. Modelo conceptual de datos (MCD).....	32

2.2. Diagrama entidad/relación (DER)	33
2.2.1. Entidad	33
2.2.2. Relación	35
2.2.3. Cardinalidad y modalidad.....	37
2.2.4. Atributos de relación	42
2.2.5. Clave primaria y claves candidatas.....	42
2.2.6. Ejemplo de diagrama entidad/relación.....	44
2.3. Diagrama entidad/relación extendido	46
Resumen.....	48
Ejercicios propuestos.....	49
Actividades de autoevaluación	50
3. EL MODELO RELACIONAL. NORMALIZACIÓN	51
Objetivos	51
Mapa conceptual del capítulo.....	52
Glosario.....	53
3.1. El modelo relacional.....	53
3.2. Normalización	54
3.2.1. Primera forma normal (1FN)	55
3.2.2. Segunda forma normal (2FN)	56
3.2.3. Tercera forma normal (3FN).....	58
3.2.4. Otras formas normales.....	59
3.3. Paso del DER al modelo físico de datos.....	59
3.3.1. Nomenclatura	60
3.3.2. Reglas de transformación	61
3.3.3. Ejemplo de transformación de DER a modelo físico de datos	70
3.4. Otras consideraciones sobre el modelo relacional.....	72
3.4.1. Índices	72
3.4.2. Vistas	73
3.4.3. Restricciones sobre campos	73
3.4.4. Integridad referencial	74
3.4.5. Usuarios y privilegios	74
3.4.6. Accesos concurrentes.....	76
3.4.7. Políticas de bloqueo	77
Resumen.....	78
Ejercicios propuestos.....	78
Actividades de autoevaluación	79
4. EL LENGUAJE SQL. DDL	81
Objetivos	81
Mapa conceptual del capítulo.....	82
Glosario.....	83
4.1. Introducción al lenguaje SQL	83
4.2. Tipos de datos en SQL	84

4.3. Lenguaje de definición de datos (DDL)	86
4.3.1. Definición de bases de datos	87
4.3.2. Definición de tablas	87
4.3.3. Definición de vistas	91
4.3.4. Definición de índices	91
4.3.5. Definición de tipos de datos	92
Resumen	93
Ejercicios propuestos	93
Actividades de autoevaluación	94
5. EL LENGUAJE SQL. DML Y DCL	95
Objetivos	95
Mapa conceptual del capítulo	96
Glosario	97
5.1. Lenguaje de manipulación de datos (DML)	97
5.1.1. La sentencia SELECT	97
5.1.2. Consulta de registros sobre una tabla	98
5.1.3. Funciones de agregación	103
5.1.4. Consulta de registros sobre varias tablas. Composiciones	107
5.1.5. Consulta de registros sobre varias tablas. Combinación de consultas	113
5.1.6. Subconsultas	114
5.1.7. Alias	116
5.1.8. Funciones integradas	118
5.1.9. Inserción de registros	119
5.1.10. Modificación de registros	121
5.1.11. Eliminación de registros	123
5.2. Lenguaje de Control de Datos (DCL)	124
5.2.1. Control de acceso a los datos	124
5.2.2. Transacciones	125
Resumen	127
Ejercicios propuestos	128
Actividades de autoevaluación	129
6. LENGUAJE DE PROGRAMACIÓN. CONSTRUCCIÓN DE GUIONES	131
Objetivos	131
Mapa conceptual del capítulo	132
Glosario	133
6.1. La necesidad de un lenguaje de programación en el SGBD	133
6.1.1. Construcción de guiones	136
6.2. Programación en SQL/PSM	136
6.2.1. Variables. Cursores	136
6.2.2. Estructuras de control de flujo	141
6.2.3. Subrutinas. Procedimientos y funciones almacenados	147
6.2.4. Control de errores. Excepciones	152
6.2.5. Triggers	155

Resumen.....	159
Ejercicios propuestos.....	159
Actividades de autoevaluación	160
7. GESTIÓN DE SEGURIDAD	161
Objetivos	161
Mapa conceptual del capítulo	162
Glosario.....	163
7.1. Introducción.....	163
7.2. Tipos de fallos	164
7.2.1. Fallos físicos	165
7.2.2. Fallos lógicos.....	165
7.3. Recuperación de fallos.....	166
7.3.1. Fallos físicos	166
7.3.2. Fallos lógicos.....	169
7.4. Copias de seguridad.....	170
7.4.1. Tipos de copias de seguridad	170
7.4.2. Estrategia básica de respaldo y recuperación	171
7.5. Transferencia de datos entre SGBD	172
Resumen.....	173
Actividades de autoevaluación	174
8. LAS BASES DE DATOS OBJETO-RELACIONALES	175
Objetivos	175
Mapa conceptual del capítulo	176
Glosario.....	177
8.1. Introducción.....	177
8.1.1. El paradigma de la orientación a objetos	178
8.1.2. Características de las bases de datos objeto-relacionales	180
8.2. Tipos de datos objeto.....	181
8.2.1. Colecciones (<i>arrays</i>).....	181
8.2.2. Tipos complejos	183
8.3. Referencias e identificadores	186
8.4. Herencia.....	188
8.4.1. De tipos	188
8.4.2. De tablas	190
8.5. Métodos.....	191
Resumen.....	193
Ejercicios propuestos.....	194
Actividades de autoevaluación	195
BIBLIOGRAFÍA	197
Publicaciones	197
Páginas web.....	198
Gráficos e imágenes	199



Introducción

En 1597 *sir* Francis Bacon acuñó la conocida frase “*Knowledge itself is a power*”, adaptada generalmente al castellano como “La información es poder”. Cuatro siglos y medio después, los padres de la informática se devanaron los sesos diseñando sistemas óptimos de almacenamiento, procedimientos de consulta y modificación de datos, mecanismos de seguridad... Como vaticinaba *sir* Francis, hoy en día la información es el activo más valioso de toda organización, y las bases de datos son la respuesta a las necesidades técnicas surgidas hace unas décadas.

Este libro trata las bases de datos relacionales y objeto-relacionales desde dos puntos de vista fundamentales: primero aborda su diseño y creación, recorriendo los pasos que hay que seguir desde la primera idea de negocio hasta su implantación funcional, combinando conceptos teóricos con actividades prácticas y apoyándose en herramientas validadas por la ingeniería del *software*, entre las que destacan la teoría de la normalización y el diagrama entidad/relación; después se centra en la gestión de la información almacenada a través del lenguaje SQL y su extensión procedimental SQL/PSM.

Durante la elaboración del texto se ha intentado no caer presa de imposiciones puntuales del mercado ni de modas pasajeras, motivo por el que la sintaxis técnica tratada, así como todos los ejemplos, corresponden a normativas internacionales, no a herramientas comerciales. Salvo donde se indique lo contrario, se utilizan las normas del estándar ISO SQL:1999, fácilmente adaptables a la sintaxis de cualquier base de datos.

Las descripciones de sentencias de los lenguajes SQL y SQL/PSM utilizan la siguiente notación, habitual en obras técnicas:

Formato	Explicación	Ejemplo
Mayúsculas	Palabra reservada del lenguaje	SELECT
Minúsculas y cursiva entre símbolos menor (<) y mayor (>)	Elemento que hay que sustituir con un valor o expresión	<lista_campos>
Texto separado por una tubería ()	Elección. Se puede utilizar el elemento a la izquierda o el elemento a la derecha de la tubería	ASC DESC
Texto entre corchetes	Opcional. Se puede omitir	[DISTINCT]
Texto entre llaves	Agrupar varios elementos separados por tuberías, indicando que se puede elegir cualquiera, pero es obligatorio elegir al menos uno	{INSERT UPDATE DELETE}
Puntos suspensivos	Más elementos del mismo tipo que el anterior	<campo1>, <campo2>, ...

Aunque el libro está destinado inicialmente a estudiantes de Ciclos Formativos de Grado Superior, la cualidad didáctica y la amplitud temática de la obra permiten su aplicación en contextos universitarios y profesionales. Con algunas ampliaciones y omisiones, los contenidos están basados en los de los módulos Gestión de Bases de Datos, del Ciclo Formativo de Grado Superior ASIR (Administración de Sistemas Informáticos en Red) y Bases de Datos, de los Ciclos Formativos de Grado Superior DAM (Desarrollo de Aplicaciones Multiplataforma) y DAW (Desarrollo de Aplicaciones Web), definidos en los siguientes documentos:

- Real Decreto 1629/2009, de 18 de noviembre, de enseñanzas mínimas del título de Técnico Superior en Administración de Sistemas Informáticos en Red.
- Real Decreto 450/2010, de 16 de abril, de enseñanzas mínimas del título de Técnico Superior en Desarrollo de Aplicaciones Multiplataforma.
- Real Decreto 686/2010, de 20 de mayo, de enseñanzas mínimas del título de Técnico Superior en Desarrollo de Aplicaciones Web.

Para complementar los contenidos de los reales decretos también se ha consultado el desarrollo del currículo de los módulos mencionados en la normativa de diversas comunidades autónomas españolas.

Desde hace diez años soy docente de la familia profesional de Informática en Ciclos Formativos de Grado Superior de Formación Profesional. Los diez años anteriores los dediqué al desarrollo de *software* en el mundo de la empresa privada, cubriendo el espectro existente entre

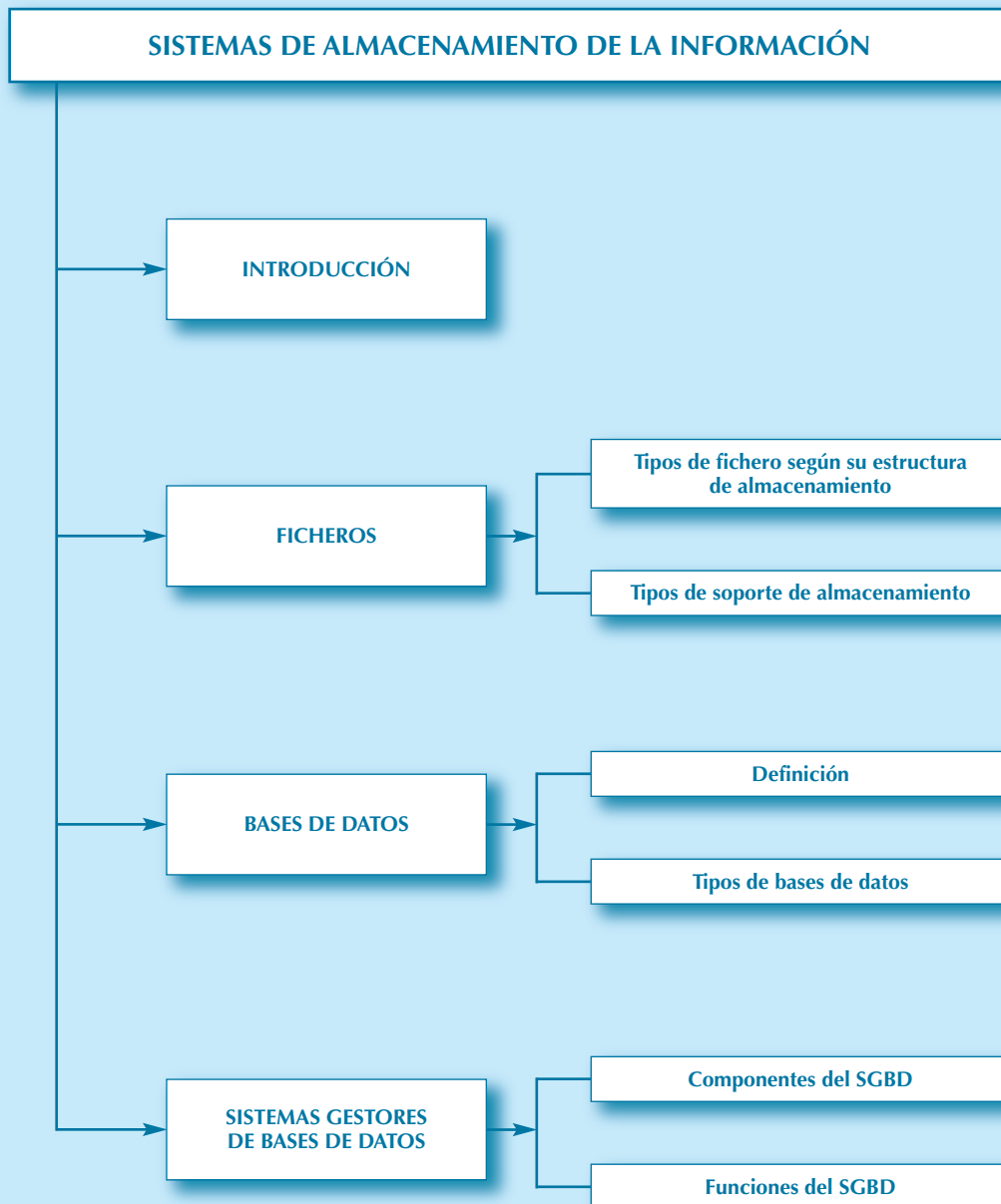
la programación y la dirección de proyectos (análisis y administración de bases de datos incluidos). Partiendo de ambos enfoques, en este libro he intentado combinar pragmatismo con fidelidad académica. Espero haber cumplido los objetivos planteados y espero, querido lector, que lo disfrute.

Sistemas de almacenamiento de la información

Objetivos

1. Comprender la problemática derivada del almacenamiento óptimo de la información.
2. Tomar cierta perspectiva histórica sobre el desarrollo de los sistemas de información.
3. Asimilar las características de los distintos tipos de fichero.
4. Entender el concepto de índice.
5. Describir los distintos modelos de bases de datos.
6. Familiarizarse con los componentes y funciones de los sistemas gestores de bases de datos.

Mapa conceptual del capítulo



Glosario

Backup. Copia de seguridad de datos o aplicaciones informáticas.

Byte. Unidad de información compuesta de ocho bits o dígitos binarios.

Fichero o archivo. Entidad consistente en secuencias de *bytes* que se almacenan en una memoria secundaria (disco duro, cinta magnética, DVD, etc.).

Interfaz de línea de comandos (Command-Line Interface, CLI). Forma de interactuar con un *software* mediante el envío de comandos por parte del usuario y en ausencia de una interfaz gráfica.

Lenguaje de marcas (markup language). Conjunto de instrucciones de codificación que permiten marcar o etiquetar diversas partes de un fichero aportando información adicional sobre su contenido, estructura o presentación.

Lenguaje de programación. Conjunto de normas lingüísticas que permiten a un ordenador comprender y ejecutar un programa.

Mainframe. Ordenador de gran potencia utilizado simultáneamente por muchos usuarios. Es habitual en entornos de banca y grandes empresas.

Networking. Área de trabajo relacionada con comunicaciones y redes de ordenadores.

Software. Programa informático.

1.1. Introducción

Según el *Diccionario* de la Real Academia Española (*DRAE*), informática es el «Conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento automático de la información por medio de ordenadores». El diccionario de Cambridge University Press define *information technology (IT)* como «la ciencia y la actividad de utilizar ordenadores y otras herramientas electrónicas para almacenar y enviar información». En ambos casos, el objeto de la disciplina es la información, y el objetivo su gestión.

Definimos *sistema de información* como el conjunto de procedimientos y funciones dirigidos a la recogida, elaboración, evaluación, almacenamiento, recuperación, condensación y distribución de informaciones dentro de una organización.

Antes de que surgieran las bases de datos el procesamiento automatizado de información se hacía mediante ficheros. Las aplicaciones eran orientadas al proceso (el esfuerzo se enfocaba al tratamiento que los datos recibían en una aplicación concreta). Los ficheros se diseñaban a medida para cada sistema de información, sin que existiera un formato común.



Figura 1.1. Un *mainframe* da servicio a muchos usuarios de forma simultánea

Esta aproximación no contemplaba la gestión de la información a medio o largo plazo. Una organización disponía de varias aplicaciones que, en algunos casos, trataban la misma información (ejemplo: el *software* utilizado por el departamento de recursos humanos debía gestionar un fichero con datos de empleados, mientras la aplicación de contabilidad mantenía otro fichero distinto con los mismos datos organizados de otra forma). Surgían los siguientes problemas:

- Redundancia de datos (duplicidad innecesaria de información).
- Mal aprovechamiento del espacio de almacenamiento.
- Aumento en el tiempo de proceso.
- Inconsistencia de información debida a la redundancia (si un dato cambiaba en el fichero de una aplicación, no cambiaba en los demás).
- Aislamiento de la información (imposibilidad de transferirla a otros programas a no ser que se desarrollara un *software* de migración específico).

Había, en definitiva, una gran falta de flexibilidad originada en la dependencia total de la estructura física de los datos.

1.2. Ficheros

Las aplicaciones gestoras de bases de datos se encargan de configurar una estructura óptima de almacenamiento de información con mínima intervención por parte del usuario. No obstante, es interesante completar la perspectiva histórica con una breve descripción teórica sobre organización de ficheros.

1.2.1. Tipos de fichero según su estructura de almacenamiento

En relación con su contenido, encontramos los siguientes tipos básicos de fichero:

- *Texto plano.* Almacenan secuencias de caracteres correspondientes a una codificación determinada (ASCII, Unicode, EBCDIC, etc.). Son legibles mediante un *software* de edición de texto como el Bloc de Notas de Windows o el Vi de Linux.

Ejemplos: los ficheros de texto con extensión .txt, los .csv de valores separados por comas, los .htm y .html correspondientes a páginas web, los de lenguajes de marcas .xml o .rss.

- *Binarios.* Contienen información codificada en binario para su procesamiento por parte de aplicaciones. Su contenido resulta ilegible en un editor de texto.

Ejemplos: archivos ejecutables (.exe), documentos de aplicaciones (.pdf, .docx, .xlsx, .pptx), ficheros de imagen, audio o vídeo (.jpg, .gif, .mp3, .avi, .mkv), archivos de sistema (.dll).

PARA SABER MÁS



El sistema de codificación de caracteres más popular es el código ASCII (*American Standard Code for Information Interchange*, código estándar estadounidense para intercambio de información), que define 256 caracteres distintos (todas las combinaciones de 8 bits, es decir, 2^8 posibilidades). Algunos de ellos, llamados caracteres de control, no representan símbolos concretos, sino que se encargan de definir acciones como el borrado, el salto de línea o el tabulador.

Cuando se utilizan ficheros de texto plano para almacenar información se pueden clasificar de acuerdo a su organización interna:

- *Secuenciales.* La información se escribe en posiciones físicamente contiguas. Para acceder a un dato hay que recorrer todos los anteriores.

```
00789521T#Paula#Sanz#González#619554687$50687452Y#José Luis#García#Viñals#
667859621$38546998X#Javier#Peinado#Martín#666932541$09653801B#Ruth#Lázaro#
Cardenal#689330247%
```

Figura 1.2. Fichero secuencial con información sobre clientes

Por cada contacto se ha decidido estructurar la información en cinco datos independientes: NIF, nombre, primer apellido, segundo apellido y número de teléfono. Nótese que en este caso el programador ha decidido utilizar la almohadilla (#) como separador de datos, el dólar (\$) como separador de contactos y el tanto por ciento (%) como marca de fin de fichero.

- *De acceso directo o aleatorio.* Cada línea de contenido se organiza con unos tamaños fijos de dato. Se puede acceder directamente al principio de cada línea.

00789521TPaula	Sanz	González	619554687
50687452YJosé Luis	García	Viñals	667859621
38546998XJavier	Peinado	Martín	666932541
09653801BRuth	Lázaro	Cardenal	689330247

Figura 1.3. Fichero de acceso directo con información sobre clientes

En esta ocasión cada contacto ocupa una línea del fichero (al final de cada una el sistema operativo incluirá uno o dos caracteres de salto de línea invisibles para el usuario), y cada dato utiliza un número de caracteres fijo, aunque no lo ocupe totalmente (en el ejemplo se reservan 15 caracteres para el nombre, aunque en el caso de Ruth solo se utilicen 4).

Como todos los clientes ocupan el mismo espacio en el fichero, podemos acceder fácilmente a cualquiera de ellos multiplicando la posición en la que se encuentra menos una por el número de caracteres que mide cada línea. Por ejemplo, si el fichero se ha creado en un sistema Windows y queremos acceder al tercer cliente, tendremos que restar uno a su posición ($3 - 1 = 2$) y multiplicar el valor resultante por la longitud de la línea (63 caracteres más los dos caracteres de salto de línea que incluye el sistema operativo, es decir, 65). Como $2 \times 65 = 130$, la información del tercer cliente se encontrará en la posición 131.

La contrapartida a esta facilidad de posicionamiento es que el tamaño del fichero crece considerablemente respecto a su versión secuencial.

- *Indexados.* Generalmente en un fichero de acceso aleatorio la información se almacena en el orden en que se da de alta. Incluso aunque se consiguiera introducir dicha información de acuerdo a algún criterio de ordenación concreto, en algunas ocasiones es útil poder ordenarla por varios criterios distintos. En el ejemplo anterior es posible que necesitemos un listado de clientes ordenado por NIF y otro por apellido. Para dar solución a este problema se creó la organización indexada, que consiste en la existencia de uno o varios archivos adjuntos que ordenan el dato (llamado clave) por el que se desea ordenar el fichero y lo relacionan con la localización de la línea correspondiente.

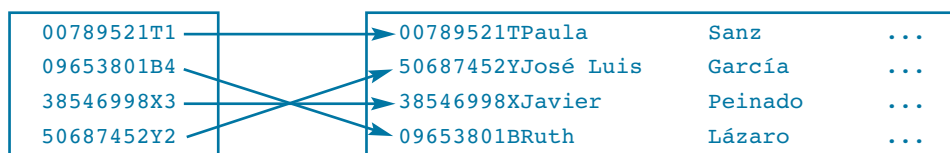


Figura 1.4. Fichero de índice por NIF de cliente y fichero de clientes original

En la figura 1.4 los NIF aparecen ordenados. Tras cada uno de ellos se ha añadido el número de línea del fichero principal donde se encuentra la información asociada. Si una aplicación *software* quisiera listar los clientes ordenados por NIF, recorrería secuencialmente el fichero de índice, y al final de cada línea encontraría la línea del fichero principal que debe leer para encontrar a cada cliente.

El siguiente fichero indexaría los clientes ordenados por su primer apellido:

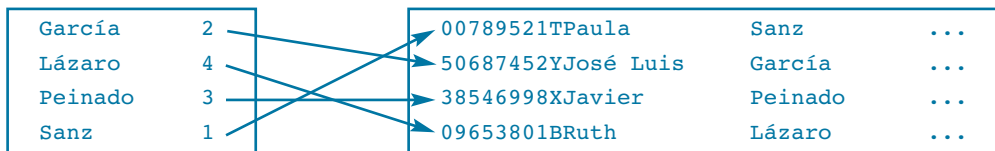


Figura 1.5

Fichero de índice por primer apellido de cliente y fichero de clientes original

Aunque se utilice en este caso para simplificar el ejemplo, generalmente el acceso a cada posición no lo marca el número de línea, sino un puntero a la celda de memoria correspondiente.

Actividad propuesta 1.1

Crear manualmente un fichero índice que ordene el ejemplo de la figura 1.3 por primer apellido, segundo apellido y nombre.



1.2.2. Tipos de soporte de almacenamiento

De acuerdo a la organización física de los datos, diferenciamos entre dos tipos de soportes:

- *Secuenciales*. Para acceder a un dato hay que recorrer todo el contenido del soporte previo a dicho dato (ejemplo: cintas magnéticas).



Figura 1.6

Cinta magnética para guardar información de respaldo

- *Direccionables*. Se puede acceder directamente a un dato sin tener que recorrer todos los anteriores (ejemplo: disco duro).



Figura 1.7

Disco duro, uno de los soportes para datos más extendidos en la actualidad

En un soporte direccionable se puede implementar un acceso secuencial, directo o indexado, mientras que en un soporte secuencial solo se podrá implementar un acceso secuencial.

1.3. Bases de datos

La evolución lógica de los problemas derivados del uso de ficheros fue estandarizar el acceso a la información, de modo que un diseño físico concreto sirviera para todas las aplicaciones de una organización. Este nuevo enfoque se centraba en los datos y no en el proceso, es decir, se estructuraba el almacenamiento de dichos datos con independencia de las aplicaciones que los fueran a utilizar. Se eliminaba la redundancia y se favorecía la transferencia de información entre aplicaciones. Aparecía el concepto de base de datos.

1.3.1. Definición

Volviendo nuevamente al *DRAE*, este nos dice que *base de datos* es un término relacionado con el mundo de la informática, y lo define como «Conjunto de datos organizado de tal modo que permita obtener con rapidez diversos tipos de información». Adoración de Miguel y Mario Piattini ofrecen una definición más precisa:

Colección o depósito de datos integrados, almacenados en soporte secundario (no volátil) y con redundancia controlada. Los datos, que han de ser compartidos por diferentes usuarios y aplicaciones, deben mantenerse independientes de ellos y su definición (estructura de la BD), única y almacenada junto con los datos, se ha de apoyar en un modelo de datos, el cual ha de permitir captar las interrelaciones y restricciones existentes en el mundo real. Los procedimientos de actualización y recuperación, comunes y bien determinados, facilitarán la seguridad del conjunto de los datos.

1.3.2. Tipos de bases de datos

Un *modelo de base de datos* es la arquitectura mediante la que se almacena e interrelaciona la información que se va a gestionar. La clasificación habitual de bases de datos toma como punto de partida el modelo subyacente:

- *Jerárquico*. Es el más antiguo. Refina la idea de fichero indexado, creando una estricta relación de jerarquía entre los datos de varios ficheros, motivo por el que presenta serias limitaciones semánticas. Relacionado con grandes máquinas (*mainframes*), su implantación comercial más conocida es IMS de IBM.
- *En red*. Introduce mejoras respecto al modelo jerárquico (mayor independencia y flexibilidad de los datos) a costa de aumentar el nivel de complejidad. Implantaciones: CODASYL, IDMS/DB de Computer Associates (actualmente CA Technologies).
- *Relacional*. Representa la información en forma de entidades y relaciones entre ellas, evitando rutas preconcebidas para localizar los datos y huyendo de la rigidez de los modelos previos. Cada entidad y cada relación aparece en forma de tablas bidimensionales (con filas y columnas). Es el modelo más extendido desde hace décadas, gracias a compañías como Oracle, IBM o Microsoft (que posteriormente evolucionaron hacia el modelo objeto-relacional), aunque hoy en día podemos encontrar bases de datos relacionales puras, como MySQL o SAP Sybase.
- *Orientado a objetos*. Aplica a los datos el paradigma de la orientación a objetos (OOP, *object-oriented programming*). Irrumpió con fuerza en los años noventa debido a las nuevas necesidades de almacenamiento de las bases de datos relacionales (imágenes, documentos, ficheros de audio y vídeo). Implantaciones: Versant, db4o, InterSystems, Objectivity.
- *Objeto-relacional*. En los últimos años los fabricantes de bases de datos relacionales han incorporado a su *software* diversas capacidades de las bases de datos orientadas a objetos, creando modelos híbridos con base relacional. Ejemplos: Oracle, Microsoft SQL Server, IBM DB2, IBM Informix, PostgreSQL.
- Otros modelos.
 - *Documental*. Destinado al almacenamiento e indexación de grandes documentos.
 - *Orientado al documento*. Gestionan datos provenientes de documentos previamente estructurados, generalmente de lenguajes de marcas (XML, JSON).
 - *Multidimensional*. Orientado al tratamiento de la información mediante algoritmos de inteligencia artificial.
 - *Deductivo*. Almacena reglas de inferencia mediante las que genera deducciones a partir de unos datos determinados.

Actividad propuesta 1.2

Con la ayuda de Internet, elaborar una lista de bases de datos comerciales y libres relacionando cada una de ellas con su modelo de datos correspondiente.



Si, en cambio, utilizamos como criterio la ubicación física de la información, podemos diferenciar entre dos grandes tipos de bases de datos:

- *Centralizadas*. La base de datos reside en una sola máquina, típicamente el servidor de base de datos.
- *Distribuidas*. La información se reparte por distintos servidores, generalmente alejados físicamente. Un ejemplo sería la base de datos de una compañía de seguros, concebida a partir de los datos de la oficina central y de los de todas sus sucursales. Su implantación exige hacer un fuerte hincapié en aspectos de *networking* y seguridad.

PARA SABER MÁS



Figura 1.8
Símbolo de base de datos

En gráficos, esquemas y literatura informática las bases de datos se representan mediante un cilindro apoyado en su base. Es una referencia a las antiguas memorias de tambor magnético, de similar aspecto físico, como se ve en el siguiente anuncio publicado en la revista *Scientific American* en 1953:

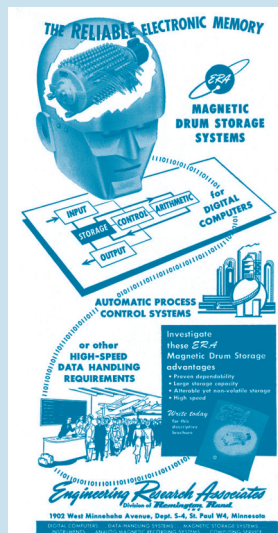


Figura 1.9
Memoria de tambor

1.4. Sistemas gestores de bases de datos

El sistema gestor de bases de datos (SGBD) es el *software* que el fabricante pone a disposición del usuario para manejar sus bases de datos. Nuevamente, De Miguel y Piattini (1993) nos definen el término con más detalle:

Un conjunto coordinado de programas, procedimientos, lenguajes, etc., que suministra, tanto a los usuarios no informáticos como a los analistas, programadores, o al administrador, los medios necesarios para describir, recuperar y manipular los datos almacenados en la base, manteniendo su seguridad.

En el mercado hay una amplia tipología de SGBD que corresponde con el modelo de base de datos subyacente.

1.4.1. Componentes del SGBD

Generalizando, podemos encontrar la siguiente enumeración de componentes en la mayoría de los SGBD:

- *Datos.* Almacenados de forma eficiente en ficheros del sistema operativo.
- *Herramientas de acceso a los datos.* Un lenguaje de programación mediante el que los usuarios técnicos puedan crear, leer y modificar la información, así como un diccionario de datos que albergue los metadatos, es decir, la información sobre el diseño de cada base de datos. Como mínimo, se ofrecerá una interfaz de línea de comandos mediante la que acceder a estas herramientas.
- *Utilidades.* Herramientas adicionales para gestión de *backups*, estadísticas, tareas programadas, mantenimiento de usuarios, grupos y permisos, etc.
- *Entornos gráficos.* Simplifican la gestión del SGBD y sirven como alternativa a la línea de comandos.

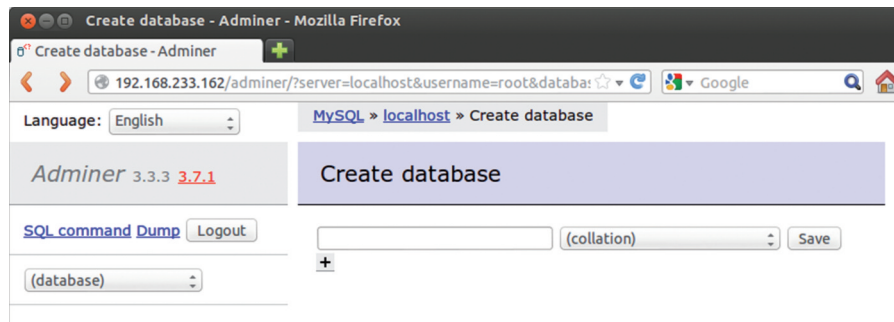


Figura 1.10. Interfaz de una base de datos

1.4.2. Funciones del SGBD

A pesar de la gran variedad de modelos y soluciones comerciales, podemos enumerar una serie de funciones comunes a un gran número de SGBD:

- Recuperar y modificar la información de los ficheros que conforman la base de datos de forma transparente para el usuario.
- Garantizar la integridad de los datos, impidiendo inconsistencias semánticas.
- Ofrecer un lenguaje de programación mediante el que interaccionar con la información.
- Proveer el diccionario de datos.
- Solucionar los conflictos derivados de accesos concurrentes a la información.
- Gestionar transacciones, garantizando la unidad de varias instrucciones de escritura relacionadas entre sí.
- Incluir utilidades de *backup*.
- Proporcionar mecanismos de seguridad para evitar accesos y operaciones indebidos.

Estos aspectos se desarrollarán en los temas siguientes.

PARA SABER MÁS



Según la consultora estadounidense Gartner, en el año 2013 la empresa Oracle Corporation contaba con la mayor cuota de mercado relativa a sistemas gestores de bases de datos con sus SGBD Oracle Database y MySQL, consolidando la tendencia de los últimos años. Tras ella se encontraban Microsoft (con SQL Server), IBM (con Informix y DB2) y SAP (con Sybase Adaptive Server Enterprise y Sybase IQ).

El *ranking* mensual de la página web www.db-engines.com estableció el siguiente índice de popularidad para gestores relacionales y objeto-relacionales en agosto de 2014:

CUADRO 1.1. SGBD(O)R ordenados por índice de popularidad

Orden	SGBD(O)R	Compañía
1	Oracle	Oracle Corporation
2	mySQL	Oracle Corporation
3	SQL Server	Microsoft Corporation
4	PostgreSQL	PostgreSQL Global Development Group
5	DB2	IBM Corporation
6	Access	Microsoft Corporation
7	SQLite	D. Richard Hipp
8	Sybase ASE	SAP Sybase
9	Teradata	Teradata
10	FileMaker	FileMaker Incorporated

Fuente: www.db-engines.com

Resumen

El almacenamiento de información ha sido uno de los grandes problemas de la informática desde sus comienzos. Inicialmente el uso de ficheros ofrecía una enorme falta de flexibilidad, creándose redundancia y dependencia de la estructura de cada archivo concreto. Si bien la inclusión de ficheros de índice aumentó considerablemente la agilidad de consulta, inserción, modificación y borrado de información, la aparición de las bases de datos dio solución al problema de forma estandarizada.

De entre los distintos modelos de base de datos, el relacional ha sido el más extendido históricamente. En la actualidad las bases de datos objeto-relacionales combinan el modelo relacional con elementos del modelo orientado a objetos.

El manejo de las bases de datos se lleva a cabo a través de los sistemas gestores de bases de datos, aplicaciones *software* que ofrecen utilidades mediante las que manipular la información a gusto del cliente.

EJERCICIOS PROPUESTOS

1. Crear un fichero de texto secuencial `agenda_secuencial.txt` con una agenda de teléfonos. Los datos que hay que incluir serán:

Nombre
Apellidos
Dirección
Ciudad
Teléfono

Incluir un mínimo de diez contactos de, al menos, cuatro ciudades distintas.

2. Partiendo de los datos del ejercicio anterior, crear un fichero de texto de acceso directo `agenda_acceso_directo.txt`. Se debe presentar una justificación del número de caracteres elegidos para cada dato.
3. Copiar el fichero `agenda_acceso_directo.txt` del ejercicio anterior a un nuevo fichero llamado `agenda_indexada.txt`. Crear un fichero de índice `agenda_apellidos_nombre.txt` que lo ordene por apellidos y nombre, y otro `agenda_ciudad.txt` que lo ordene por ciudad.
4. Borrar un contacto cualquiera de `agenda_indexada.txt` e insertar un contacto nuevo, actualizando a la vez los ficheros de índice asociados.
5. Consultar en Internet información sobre las prestaciones de los SGBD comerciales y libres más utilizados en la actualidad.

ACTIVIDADES DE AUTOEVALUACIÓN

1. Un archivo de imagen .png es un fichero:
 - ☐ a) Binario.
 - ☐ b) Plano.
 - ☐ c) Ninguno de los dos.
2. Un fichero secuencial se puede implementar:
 - ☐ a) Sobre un soporte secuencial.
 - ☐ b) Sobre un soporte direccionable.
 - ☐ c) Sobre ambos.
3. Respecto a los ficheros secuenciales y de acceso directo, los ficheros indexados son más rápidos:
 - ☐ a) En inserciones y modificaciones de datos.
 - ☐ b) En borrado de datos.
 - ☐ c) En búsqueda de datos.
4. El SGBD comercial Oracle obedece al modelo:
 - ☐ a) Relacional.
 - ☐ b) Objeto-relacional.
 - ☐ c) Orientado a objetos.
5. Un SGBD:
 - ☐ a) Debe ofrecer herramientas gráficas para acceder a la información.
 - ☐ b) Debe ofrecer al menos una línea de comandos desde la que acceder a la información.
 - ☐ c) Se gestiona con herramientas externas.

SOLUCIONES:

1. ☒ a ☐ b ☐ c

2. ☐ a ☐ b ☒ c

3. ☐ a ☐ b ☒ c

4. ☐ a ☒ b ☐ c

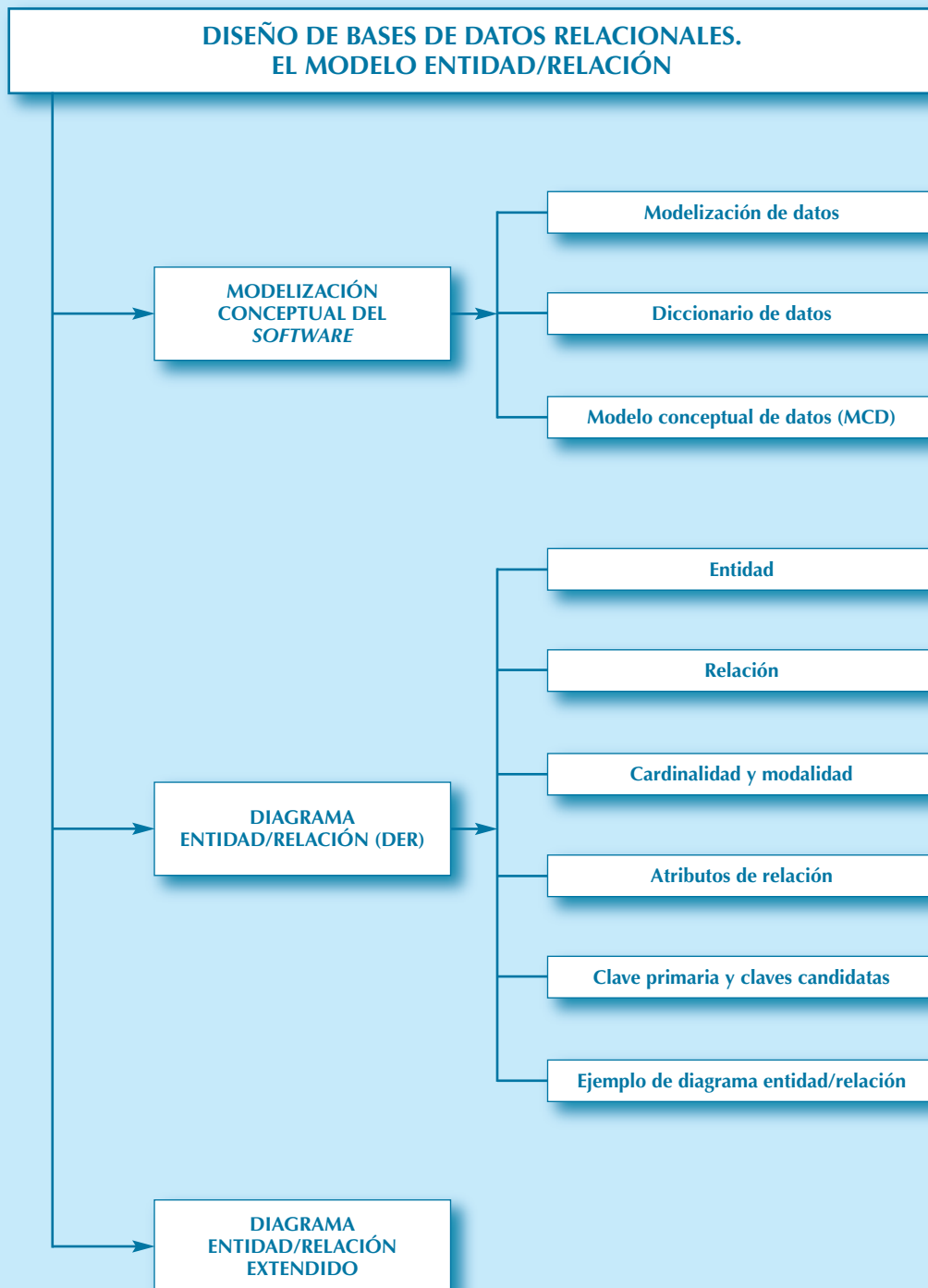
5. ☐ a ☒ b ☐ c

Diseño de bases de datos relacionales. El modelo entidad/relación

Objetivos

1. Ubicar el diseño de bases de datos relacionales en el marco del análisis de desarrollo del *software*.
2. Familiarizarse con el concepto de diccionario de datos.
3. Comprender la utilidad del diagrama entidad/relación como herramienta primordial en el diseño de bases de datos relacionales.
4. Dominar los conceptos y la terminología asociados al diagrama entidad/relación.
5. Describir apropiadamente la cardinalidad y modalidad de una relación.
6. Conocer los elementos que definen el diagrama entidad/relación extendido.

Mapa conceptual del capítulo



Glosario

Análisis (de requisitos del software). Proceso sistemático orientado a obtener los requisitos que debe cumplir un *software* (necesidades del usuario, comportamiento interno y externo, documentación y evolución del sistema).

ANSI. Instituto Nacional Estadounidense de Estándares (American National Standards Institute). Organismo de estandarización.

Diagrama de flujo de datos (DFD). Técnica que representa gráficamente los límites de un sistema de información, mostrando el flujo de datos entre sus integrantes y ofreciendo una descomposición de procesos a varios niveles.

Diagrama de transición de estados (DTE). Herramienta gráfica que representa los distintos estados que puede tomar cada parte de un sistema de información, así como los eventos que propician el cambio entre estados.

Herramienta CASE (*computer-aided software engineering*, ingeniería del software asistida por ordenador). Aplicación que presta soporte a diversas fases del desarrollo de *software* aportando un enfoque de ingeniería.

Ingeniería del software. Aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del *software* (definición del Instituto de Ingeniería Eléctrica y Electrónica –IEEE, Institute of Electrical and Electronics Engineers–, 1993).

Metodología (de desarrollo de software). Conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas de información (definición de Richard Maddison, 1983).

2.1. Modelización conceptual del software

Cualquier desarrollo de un sistema de información parte de un proceso de análisis estructurado cuyo objetivo es identificar de forma precisa, exhaustiva y verificable las necesidades del sistema. Una vez que la abstracción conceptual del problema tome forma en papel, se procederá a implantarla como solución informatizada.

Roger S. Pressman propone un modelo de análisis destinado a cumplir estos objetivos. Centrado en el contexto relacional y tomando el diccionario de datos como centro de la arquitectura, divide la problemática en tres ámbitos a los que asocia herramientas gráficas de representación:

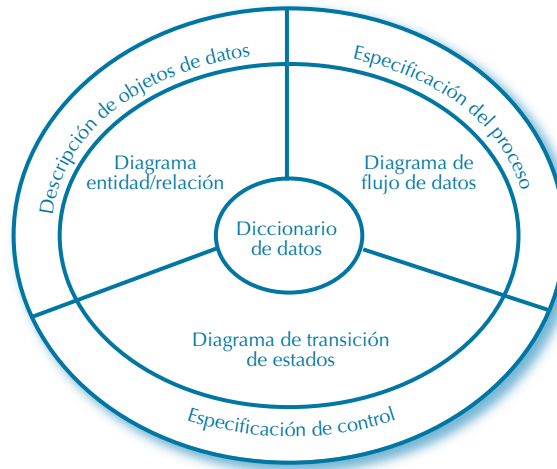


Figura 2.1
Modelo de análisis
de Roger S. Pressman

- *Descripción de objetos de datos.* Representa dichos objetos y las relaciones entre ellos. Utiliza el diagrama entidad/relación (DER).
- *Especificación de proceso.* Identifica las distintas funcionalidades del sistema y sus relaciones, así como la forma en que la información se transforma a su paso por ellas. Se expresa en el diagrama de flujo de datos (DFD).
- *Especificación de control.* Muestra los distintos estados en los que se puede encontrar el sistema y las transiciones de uno a otro gracias al diagrama de transición de estados (DTE).

Nuestra problemática nos lleva a analizar de cerca el primero de estos tres ámbitos.

PARA SABER MÁS



El proceso de análisis es fundamental en todo desarrollo de *software* (incluyendo, por supuesto, el diseño y creación de la base de datos). Un error habitual consiste en empezar a implantar el sistema de información sin haber dedicado el tiempo adecuado a reflexionar sobre los requisitos, necesidades y otros aspectos de planificación. Según la ingeniería del *software*, lo ideal es dedicar más tiempo al análisis del que requiere el trabajo técnico de implantación. Por ese motivo se debe considerar que las herramientas más poderosas de todo informático son el lápiz y el papel.

2.1.1. Modelización de datos

La modelización de datos es una actividad que se realiza a lo largo del proceso de desarrollo del sistema de información en varias fases:

1. Al principio del proceso de análisis se crea el modelo conceptual de datos.
2. Dicho modelo se describe en términos gráficos mediante el diagrama entidad/relación, conformando un modelo lógico de datos.

3. Durante el proceso de diseño del *software* el diagrama entidad/relación se refina, convirtiéndose en un modelo físico de datos.
4. Finalmente, dicho modelo físico se implanta en el SGBD elegido, adaptándose a la sintaxis y características de rendimiento de este.

Las ventajas de la modelización de datos derivan de la noción de independencia. Cualquier cambio relativo a nuevos requisitos o flujos de información se podrá plasmar con independencia de las herramientas que se utilicen en la implantación final, facilitando el mantenimiento del sistema.

2.1.2. Diccionario de datos

El diccionario de datos, también llamado metabase, almacena metadatos, es decir, datos sobre los datos. Es la herramienta a la que acuden los desarrolladores de *software* cuando necesitan información sobre los distintos elementos de una base de datos. Se encarga de la categorización lógica de los datos, incluyendo descripciones, significado, estructura y consideraciones de edición, uso y seguridad de dichos datos.

```

...
Tabla: TSocio
Campos:
    cNIF          CHAR(9)          NOT NULL UNIQUE
    cNombre       VARCHAR(30)       NOT NULL
    cApellidos    VARCHAR(60)       NOT NULL
    cDireccion    VARCHAR(100)
    cTelefono     CHAR(9)           NOT NULL
    dNacimiento   DATE              NOT NULL
    dAlta         DATE              NOT NULL
Clave primaria: cNIF
Claves ajenas:
    N/A
Índices:
    iSocio_PK          cNIF ASC
    iSocio_Apellidos_Nombre cApellidos ASC + cNombre ASC
    iSocio_Alta        dAlta DESC
-----
Tabla: TPrestamo
Campos:
    cSignatura     VARCHAR(15)     NOT NULL
    cNIF           CHAR(9)         NOT NULL
    dPrestamo      DATE            NOT NULL
Clave primaria: cSignatura ASC + cNIF ASC + dPrestamo DESC
Claves ajenas:
    cSignatura → TEjemplar.cSignatura
    cNIF       → TSocio.cNIF
Índices:
    iPrestamo_PK          cSignatura ASC + cNIF ASC + dPrestamo DESC
    iPrestamo_FK_NIF      cNIF ASC
    iPrestamo_FK_Signatura cSignatura ASC
-----
...

```

Figura 2.2. Ejemplo de un fragmento de un diccionario de datos

2.1.3. Modelo conceptual de datos (MCD)

El modelo conceptual de datos (MCD) representa la visión estática del dominio de la información del problema, y permite identificar la estructura estática de las entidades de datos y las relaciones existentes entre ellas.

Características del MCD:

1. Debe albergar el universo del discurso, es decir, toda la información que ha de manejar el sistema.
2. Ha de representar el estado final al que pueden llegar los datos.
3. Cualquier cambio en el sistema de información se debe reflejar en el modelo de datos y viceversa.

La herramienta más poderosa para llevar a cabo la modelización conceptual de datos es el diagrama entidad/relación, una técnica propuesta por Peter Chen en 1976 que representa el conjunto de datos del sistema de información de forma estandarizada y universal.

PARA SABER MÁS



El organismo ANSI definió, a través de su grupo X3/SPARC, tres niveles abstractos relacionados con tres esquemas correspondientes que dan concreción a dichos niveles en una base de datos. Son los siguientes:

- Nivel interno. Forma en que se pueden representar los datos en un soporte de almacenamiento secundario. Su esquema asociado es el conjunto de ficheros sobre los que se almacena la base de datos y la estructura de dichos ficheros.
- Nivel externo. Forma en que usuarios y aplicaciones ven y acceden a la información contenida en la base de datos. Su esquema asociado son los datos a los que un usuario o aplicación pueden acceder en pertenencia o visibilidad. Se materializa, entre otros, en vistas.
- Nivel conceptual. Conjunto de utilidades, objetos y relaciones entre los mismos que dan soporte al sistema de información representado en la base de datos. Su esquema asociado queda patente en el diccionario de datos y en el modelo entidad/relación.

Al margen de lo que dicta el estándar, se suele hablar de un cuarto nivel llamado nivel canónico, correspondiente a las restricciones y particularidades que el fabricante del gestor introduce para hacer el dato propietario. Es una especie de supernivel que opera sobre los otros tres, y que explica por qué la transferencia de información entre distintos SGBD suele necesitar de un complejo proceso de migración, como se verá en el tema 7.

2.2. Diagrama entidad/relación (DER)

También conocido como diagrama o modelo E-R, E/R o entidad/asociación, el diagrama entidad/relación, según Métrica-3, es

una técnica cuyo objetivo es la representación y definición de todos los datos que se introducen, almacenan, transforman y producen dentro de un sistema de información, sin tener en cuenta las necesidades de la tecnología existente, ni otras restricciones.

Merece la pena recalcar ese factor de independencia con respecto a la implementación final. El DER dará solución al problema planteado sin importar cuál sea el SGBD comercial que se vaya a utilizar. Para ello parte de una serie de conceptos abstractos que se detallan a continuación.

PARA SABER MÁS



Métrica-3 es la versión 3 de la Metodología de Planificación, Desarrollo y Mantenimiento de Sistemas de Información de la Administración Pública de España. Distintos Gobiernos han definido metodologías de desarrollo informático propias, como Merise en Francia o SSADM en el Reino Unido.

2.2.1. Entidad

Una entidad es cualquier objeto real o abstracto que tiene existencia por sí mismo y se puede identificar de una forma clara y precisa, y del cual se desea registrar información en el sistema. Es el elemento fundamental que hay que caracterizar. Se representa con sustantivos en singular que encierran un concepto, y es labor del analista identificar dichos sustantivos. Ejemplos de entidad: “Empleado”, “Cliente”, “Factura”, “Línea de factura”, “Proveedor”.

Entidad es un concepto abstracto. Cada elemento concreto de una entidad es una *ocurrencia*. En el ejemplo de la entidad “Empleado”, cada uno de los empleados es una ocurrencia de dicha entidad.

A su vez, cada ocurrencia presenta una serie de datos asociados. Un empleado tendrá nombre, apellidos, NIF, dirección postal, número de teléfono, etc. Cada uno de esos datos es un *atributo*, y cada ocurrencia tiene distintos valores para cada atributo (en la entidad “Empleado”, un empleado concreto tendrá “Juan Antonio” como valor de su atributo “Nombre”, “García Corredor” como valor de su atributo “Apellidos” y “52874660Y” como valor de su atributo “NIF”).

Toda entidad debe cumplir dos características:

- Presencia del mismo conjunto de atributos para todas las ocurrencias, independientemente de que alguna ocurrencia carezca de valor para algún atributo.
- Diferenciación unívoca de ocurrencias. No puede haber dos ocurrencias con los mismos valores para todos sus atributos.

La representación gráfica de una entidad consiste en un rectángulo con el nombre de la entidad en su interior, generalmente en mayúsculas. Los atributos asociados a una entidad se pueden representar de dos formas: mediante óvalos que incluyen el nombre del atributo o mediante círculos con el nombre del atributo en el exterior:

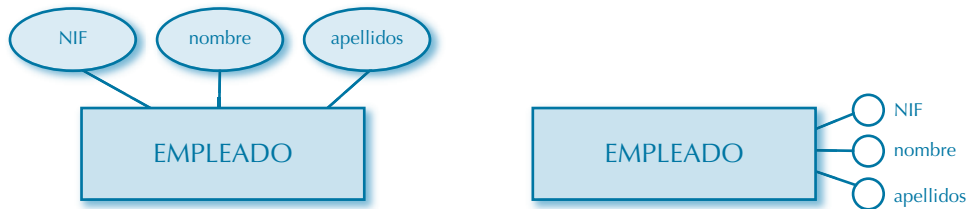


Figura 2.3. Representaciones gráficas de entidades y atributos

Hay dos tipos de entidades:

- *Fuerte o regular.* No depende de otra. Corresponde a la inmensa mayoría de las entidades.
- *Débil.* La existencia de sus ocurrencias depende de la existencia de ocurrencias en otras entidades.

Supongamos un sistema de información de una biblioteca. Si queremos llevar un control sobre las multas impuestas a los socios por devolución tardía de libros, crearemos una entidad “Multa”. Toda multa se impone a un socio; por tanto, no podrá haber ocurrencias en la entidad “Multa” si no existe una ocurrencia correspondiente a un socio en la entidad “Socio”.

Para que una entidad sea débil ha de serlo respecto a todas las entidades con las que se relaciona. Las entidades débiles se representan mediante un doble rectángulo:



Figura 2.4. Representación gráfica de una entidad débil



Actividad propuesta 2.1

Elaborar una lista de posibles entidades de una aplicación de gestión de la caja registradora de un bar.

2.2.2. Relación

Una relación es una asociación o vínculo entre ocurrencias de varias entidades. Se nombran con expresiones verbales. Ejemplos de relaciones serían la existente entre las ocurrencias de la entidad “Cliente” y las de la entidad “Factura” (ya que toda factura corresponde a un cliente), y a la que podríamos llamar “genera” (se leería “cliente genera factura”).

La representación gráfica de una relación consiste en un rombo rodeando su nombre:



Figura 2.5. Representación gráfica de una relación

De acuerdo al número de entidades cuyas ocurrencias relacionan, podemos dividir las relaciones en varias categorías:

- *Binarias.* Relacionan entre sí ocurrencias de dos entidades.
Continuando con el ejemplo anterior, esta sería la representación gráfica de la relación entre “Cliente” y “Factura”:



Figura 2.6. Relación binaria

- *Ternarias.* Relacionan entre sí ocurrencias de tres entidades.
Imaginemos ahora un sistema de información correspondiente a un centro de enseñanza. Cada profesor imparte una serie de asignaturas a distintos grupos de alumnos. Si disponemos de las entidades “Profesor”, “Grupo” y “Asignatura”, tendríamos que relacionar sus ocurrencias mediante la siguiente relación ternaria:

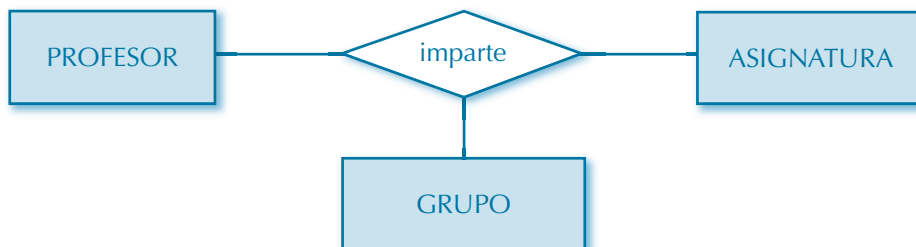


Figura 2.7. Relación ternaria

- *N-arias*. Dependiendo de la complejidad de nuestro modelo de datos podemos relacionar ocurrencias de más de tres entidades. Basta suponer que en la relación del ejemplo anterior sea necesario, además, indicar en qué aula se imparte cada asignatura a cada grupo:

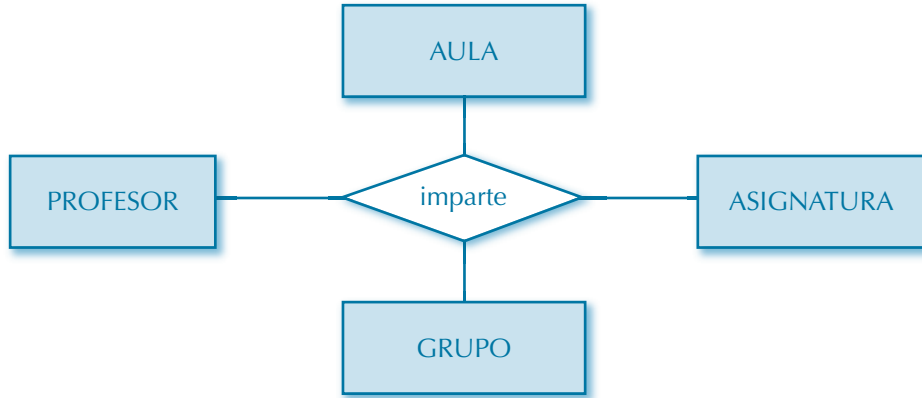


Figura 2.8. Relación cuaternaria

Como criterio de diseño se identifican relaciones ternarias o n -arias cuando la acción identificada por la relación afecta de forma simultánea a las ocurrencias de todas las entidades relacionadas (un profesor imparte una asignatura a un grupo en un aula en un momento concreto, por lo que existe simultaneidad temporal).

- *Reflexivas*. Relacionan entre sí ocurrencias de la misma entidad. Para comprenderlas correctamente hay que tener en cuenta que las ocurrencias que relacionan, aun perteneciendo a la misma entidad, están jugando papeles distintos.

En el siguiente ejemplo hemos de visualizar una entidad “Empleado” que alberga ocurrencias de empleados de una empresa. Hay empleados que son jefes de otros, y todos los empleados figuran como ocurrencias de la entidad “Empleado”. En este caso, debemos relacionar ocurrencias de “Empleado” correspondientes a empleados que tienen jefe con ocurrencias de “Empleado” correspondientes a empleados que son jefes de otros:

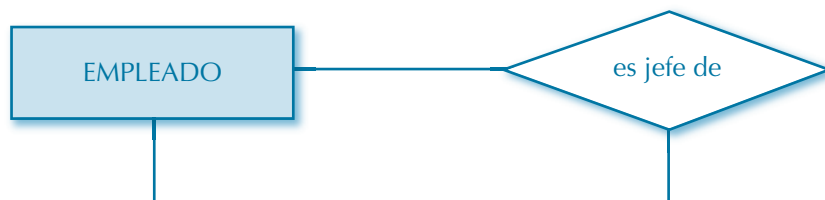


Figura 2.9. Relación reflexiva

PARA SABER MÁS



A partir de la notación original definida por Peter Chen se desarrollaron nuevos tipos de representación gráfica bastante extendidos tanto en el ámbito profesional como en el académico. Merece la pena nombrar las notaciones de Charles Bachman, Richard Barker y, muy especialmente, la del británico James Martin, que se abordará en el próximo tema como refinación del diagrama entidad/relación.

Actividad propuesta 2.2

Listar las posibles relaciones entre las entidades resultantes de la actividad propuesta 2.1. Debatir cuáles son candidatas a ser reflexivas, ternarias o n -arias.



2.2.3. Cardinalidad y modalidad

Para dotar de contenido semántico a una relación hay que especificar de qué modo se relacionan entre sí las ocurrencias de las distintas entidades, estableciendo ámbitos, límites y restricciones.

La *cardinalidad* (llamada *tipo de correspondencia* por algunos autores) indica el número máximo de ocurrencias de una entidad con las que se puede relacionar una ocurrencia de otra entidad. Es posible que en nuestro sistema de información toda factura deba emitirse a nombre de un solo cliente, pero que cada cliente pueda emitir muchas facturas; puede que cada proveedor suministre varios artículos y que cada artículo sea suministrado por varios proveedores; y quizás cada empleado se sienta en una ubicación concreta de la oficina y a cada ubicación corresponda solamente un empleado. La cardinalidad refleja esta casuística del siguiente modo:

- $1:N$ (uno a ene/uno a muchos). Una ocurrencia de una entidad puede relacionarse con varias de otra entidad, pero cada ocurrencia de la segunda entidad solo puede relacionarse con una única ocurrencia de la primera entidad.

Supongamos una empresa donde cada empleado pertenece a un departamento y en cada departamento puede haber varios empleados. Esta sería la representación gráfica:

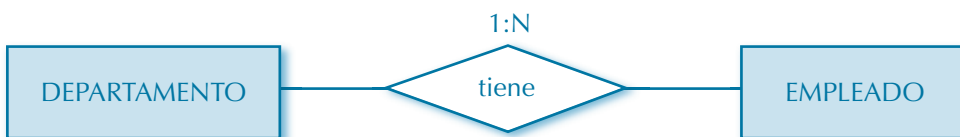


Figura 2.10. Relación 1:N

- *M:N* (eme a ene/muchos a muchos). Cada ocurrencia de una entidad puede relacionarse con varias de otra entidad, y cada ocurrencia de la segunda entidad también puede relacionarse con varias de la primera.

Si en nuestro sistema de información un músico puede tocar varios instrumentos y un instrumento puede ser tocado por varios músicos:

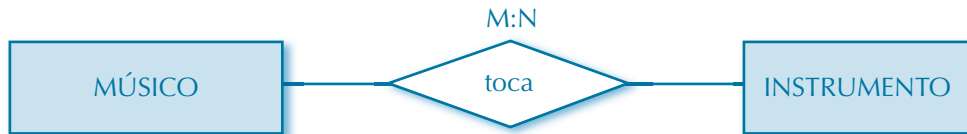


Figura 2.11. Relación *M:N*

- *1:1* (uno a uno). Una ocurrencia de una entidad se relaciona con otra ocurrencia de otra entidad y viceversa.

Una consultora financiera podría asignar a cada cliente una única cartera de inversión propia:

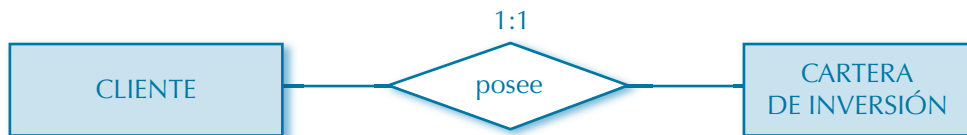


Figura 2.12. Relación *1:1*



Actividad propuesta 2.3

Completar la actividad propuesta 2.2 incluyendo la cardinalidad de cada relación.

La cardinalidad delimita los límites superiores de una relación, pero no define su obligatoriedad. ¿Todo instrumento debe ser tocado por algún músico? ¿Podría haber alguna ocurrencia en la entidad “Instrumento” que no tuviera relación con otra ocurrencia en la entidad “Músico”? Puede que, como requisito del sistema de información, la entidad “Instrumento” deba contener una lista de instrumentos independientemente de que haya algún músico que los toque (quizás una de las ocurrencias de “Instrumento” corresponda al acordeón y, en cambio, en “Músico”

no haya ninguna ocurrencia correspondiente a un músico que toque el acordeón). Pero también puede que el sistema de información exija que las ocurrencias de “Instrumento” se carguen a medida que se introduzcan en “Músico” ocurrencias de músicos, de modo que todos los instrumentos tendrían su correspondencia con, al menos, un músico. Obviamente, las dos situaciones son incompatibles en un buen diseño de datos, y la cardinalidad no nos aporta la información necesaria para solventar esta disyuntiva. Hemos de abordar un nuevo concepto.

La *modalidad* (llamada por algunos autores *cardinalidad*, con la consiguiente confusión) define el número mínimo y máximo de ocurrencias de una entidad que pueden estar relacionadas con una ocurrencia de otra u otras entidades, identificando relaciones optativas (en las que no tiene por qué haber correspondencia). La modalidad se indica a ambos lados de la relación, y su valor máximo coincide con el valor de la cardinalidad correspondiente al lado de la relación en el que nos encontremos. Puede ser de los siguientes tipos:

- *0..1* (cero a uno). Cada ocurrencia de la primera entidad puede relacionarse con una ocurrencia de la segunda entidad o no. No puede relacionarse con varias.
- *1..1* (uno a uno). Cada ocurrencia de la primera entidad debe relacionarse obligatoriamente con una y solo una ocurrencia de la segunda entidad.
- *1..N* (uno a ene/uno a muchos). Cada ocurrencia de la primera entidad debe relacionarse obligatoriamente con al menos una ocurrencia de la segunda entidad. Puede relacionarse con varias.
- *0..N* (cero a ene/cero a muchos). Cada ocurrencia de la primera entidad no tiene limitada su relación con ocurrencias de la segunda entidad. Puede relacionarse con una, varias o ninguna.

Veamos el siguiente ejemplo, correspondiente a un concesionario de automóviles:

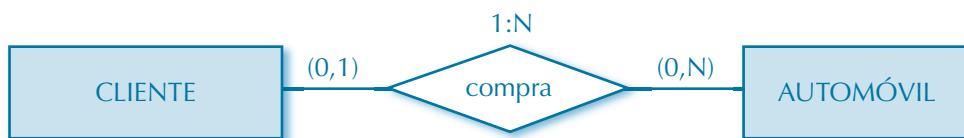


Figura 2.13. Relación 0..1 a 0..N

Para cada entidad de la relación debemos leer su modalidad en el lado opuesto. En este caso, la modalidad de “Cliente” con respecto a “Automóvil” es de 0..N, y la de “Automóvil” con respecto a “Cliente” es de 0..1. Esta relación “compra” se leería del siguiente modo:

- Toda ocurrencia de “Cliente” puede relacionarse con varias ocurrencias de “Automóvil”, con una o con ninguna (en lenguaje natural: un cliente puede no comprar, comprar un automóvil o comprar varios automóviles).
- Toda ocurrencia de “Automóvil” puede relacionarse con una sola ocurrencia de “Cliente” o no (un automóvil puede ser vendido a un solo cliente, o bien no ser vendido; en cualquier caso, el mismo automóvil no puede ser vendido a varios clientes).

Este otro ejemplo refleja una situación diferente:

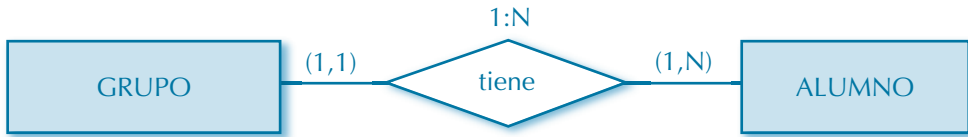


Figura 2.14. Relación 1..1 a 1..N

Al igual que en el caso anterior, la cardinalidad es 1:N, pero la modalidad es más restrictiva (1..N de “Grupo” a “Alumno” y 1..1 de “Alumno” a “Grupo”):

- Toda ocurrencia de “Grupo” está obligada a relacionarse con al menos una ocurrencia de “Alumno”, pudiendo relacionarse con varias (todo grupo de alumnos debe tener alumnos).
- Toda ocurrencia de “Alumno” está obligada a relacionarse con una y solo una ocurrencia de “Grupo” (cada alumno debe pertenecer de forma obligatoria a un grupo y solamente a uno).

Analicemos ahora un caso de modalidad sobre una cardinalidad de M:N. Imaginemos un sistema de información de una biblioteca donde a los socios se les da de alta en el momento en que retiran su primer libro en préstamo (por tanto, en el sistema no puede haber socios a los que no se haya prestado un libro):



Figura 2.15. Relación 0..N a 1..N

La interpretación sería:

- Toda ocurrencia de “Socio” está obligada a relacionarse con al menos una ocurrencia de “Libro”, y puede relacionarse con varias (todo socio debe haber recibido al menos un libro en préstamo).
- Toda ocurrencia de “Libro” puede relacionarse con varias ocurrencias de “Socio”, con una o con ninguna (un libro puede haber sido prestado a un socio, a varios o a ninguno).

Nótese que, una vez que se utiliza el concepto de modalidad, no es necesario indicar la cardinalidad, ya que esta se puede deducir a partir de aquella (tomando los valores máximos a cada lado de la relación).

En las relaciones ternarias y n -arias la modalidad se lee de una forma algo distinta. La indicación de modalidad correspondiente a cada entidad indica cómo se relacionan las ocurrencias de dicha entidad con una vinculación de ocurrencias del resto de las entidades participantes en la relación. Veamos el siguiente ejemplo:

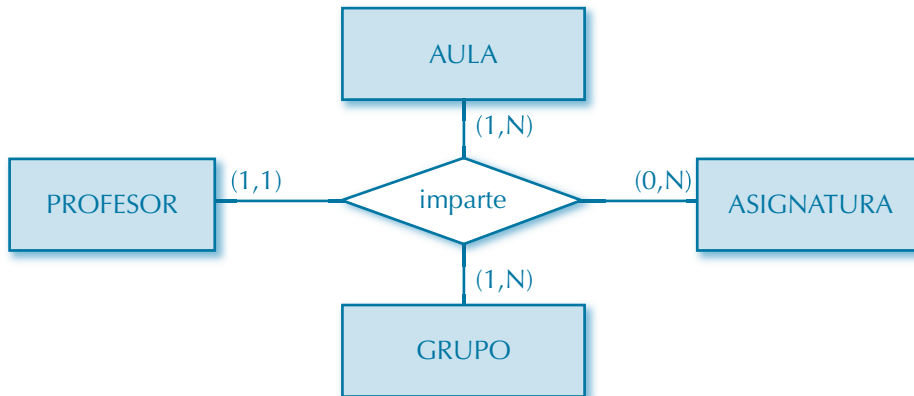


Figura 2.16. Relación cuaternaria

El 1..N de “Aula” indica que la combinación profesor-grupo-asignatura puede ocurrir en una o varias aulas (un profesor puede impartir una asignatura a un grupo en un aula o en varias —obviamente, en momentos diferentes en el tiempo—). El 0..N de “Asignatura” nos dice que por cada ocurrencia de la relación profesor-grupo-aula puede haber una, varias o ninguna asignatura (un profesor puede impartir varias asignaturas al mismo grupo en la misma aula, pero también puede reunirse en un aula con un grupo para llevar a cabo una tarea que no sea impartir una asignatura —por ejemplo, una reunión de tutoría—). El 1..N de “Grupo” implica que todo profesor que imparte una asignatura en un aula la puede impartir a uno o varios grupos (es decir, puede impartir la misma asignatura a varios grupos en la misma aula), y el 1..1 de “Profesor” quiere decir que solamente un profesor y solamente uno puede impartir la misma asignatura al mismo grupo en la misma aula.

Actividad propuesta 2.4

Escribir la modalidad de las relaciones aportadas como solución a la actividad propuesta 2.3.



2.2.4. Atributos de relación

En algunas ocasiones, ciertas propiedades o características susceptibles de constituir un atributo afectan a más de una entidad. Retomando el ejemplo de la figura 2.15, es fácil identificar posibles atributos de la entidad “Socio” (“NIF”, “Nombre”, “Apellidos”, “Dirección”, “Teléfono”, “Fecha de nacimiento”, “Fecha de alta”) y de la entidad “Libro” (“Signatura”, “Título”, “Autor”, “Editorial”, “Año de publicación”). No obstante, un dato importante que el sistema debe registrar es la fecha del préstamo. No puede figurar como atributo de “Socio”, porque impediría que un socio pudiera tomar más de un libro en préstamo, y no puede figurar como atributo de “Libro” porque impediría que un libro fuera prestado más de una vez. Afecta a ocurrencias de ambas entidades, por lo que debe figurar en la relación:

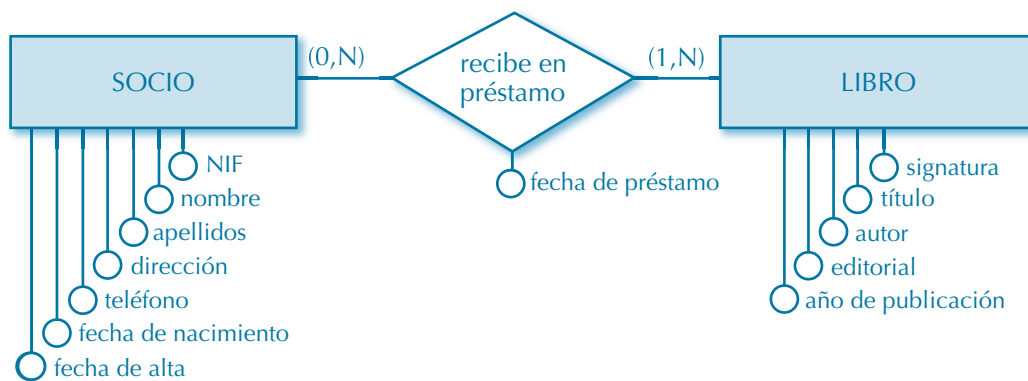


Figura 2.17. Atributo de relación



Actividad propuesta 2.5

Partiendo de la figura 2.17, decidir si los siguientes atributos serían de relación o corresponderían a alguna de las entidades:

- Fecha de adquisición del libro por parte de la biblioteca.
- Número de libros adquiridos por el socio en el momento del préstamo.
- Correo electrónico del socio.

2.2.5. Clave primaria y claves candidatas

En toda entidad hay al menos un atributo o conjunto de atributos cuyos valores identifican de forma unívoca a cada una de las ocurrencias. Observemos el siguiente ejemplo:

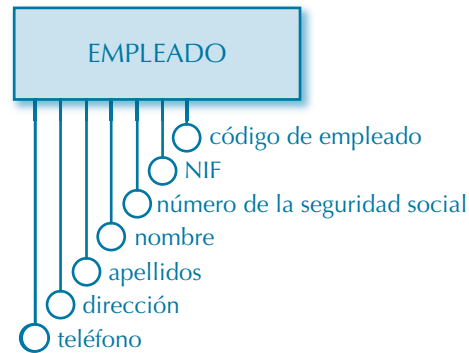


Figura 2.18
Entidad con atributos

Si queremos elegir un conjunto de atributos que identifiquen de forma única a cada empleado, podemos pensar en la combinación de “Nombre” y “Apellidos”, pero presenta un gran problema: no garantiza la unicidad (puede haber dos empleados llamados “Juan Pérez García”). Si seguimos buscando entre la lista de atributos, veremos tres posibles opciones: “Código de empleado”, “NIF” y “Número de la Seguridad Social”. Cada uno de los tres sirve para identificar a cada ocurrencia de forma única e inconfundible (no puede haber dos empleados con el mismo NIF, ni con el mismo número de la Seguridad Social, ni con el mismo código de empleado, suponiendo que dicho código haya sido definido de forma única). Esos tres atributos son llamados *claves candidatas*, y el que elijamos para identificar a las ocurrencias de la entidad será la *clave primaria* (*primary key*). Por motivos de rendimiento de la base de datos resultante, se recomienda elegir la clave primaria más pequeña posible.

En un diagrama entidad/relación las claves primarias se representan subrayando el nombre del atributo si se ha optado por dibujar los atributos en el interior de un óvalo, u oscureciendo el círculo si se ha decidido utilizar el otro tipo de representación gráfica. Supongamos que hemos elegido “NIF” como clave primaria:

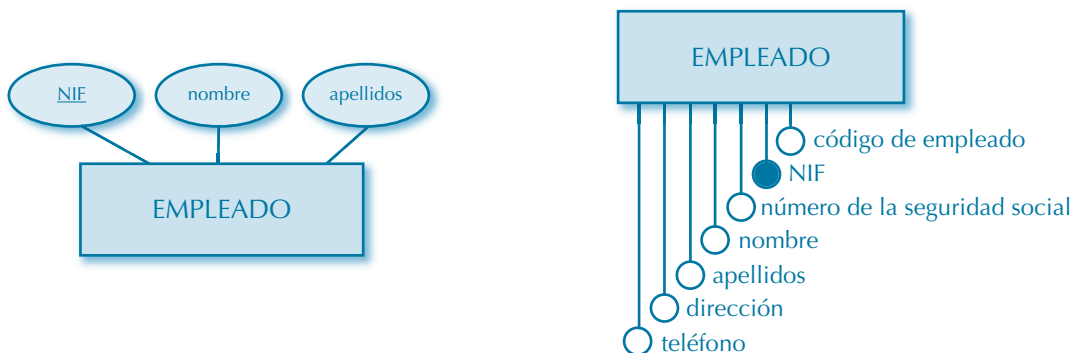


Figura 2.19. Representaciones gráficas de clave primaria

A veces un solo atributo no es suficiente para identificar de forma unívoca todas las ocurrencias de una entidad, por lo que hay que construir una clave primaria que incluya los valores de varios atributos en el orden especificado, es decir, una *clave primaria compuesta*. En el siguiente ejemplo, la entidad “Cuenta bancaria” necesita cuatro atributos para definir su clave primaria, ya que cada uno de ellos es susceptible de repetir valores en distintas ocurrencias:

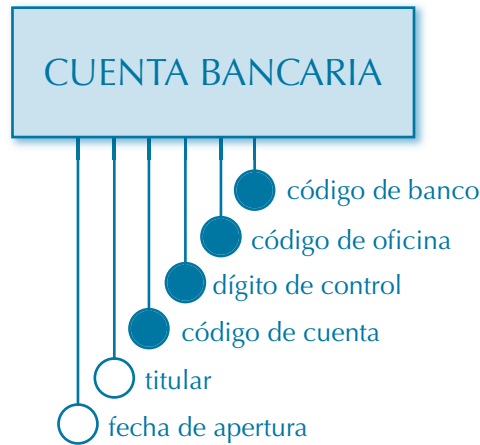


Figura 2.20. Clave primaria compuesta



Actividad propuesta 2.6

Elegir la clave primaria y las claves candidatas, si las hubiera, de la entidad “Libro” presente en la figura 2.17.

2.2.6. Ejemplo de diagrama entidad/relación

Se ha de definir el modelo de datos de una biblioteca con las siguientes restricciones básicas:

- Cada libro registrado cuenta con varios ejemplares físicos.
- Todo libro corresponde a una o varias temáticas, está escrito por uno o varios autores y publicado por una sola editorial.
- Los ejemplares se prestan a los socios. Hay que guardar la fecha de préstamo.
- Es necesario almacenar la nacionalidad o nacionalidades de cada autor, así como el país donde cada editorial tiene su sede principal.

A pesar de que falta mucha información sobre los requisitos de la aplicación, el siguiente diagrama cumple con las restricciones expuestas:

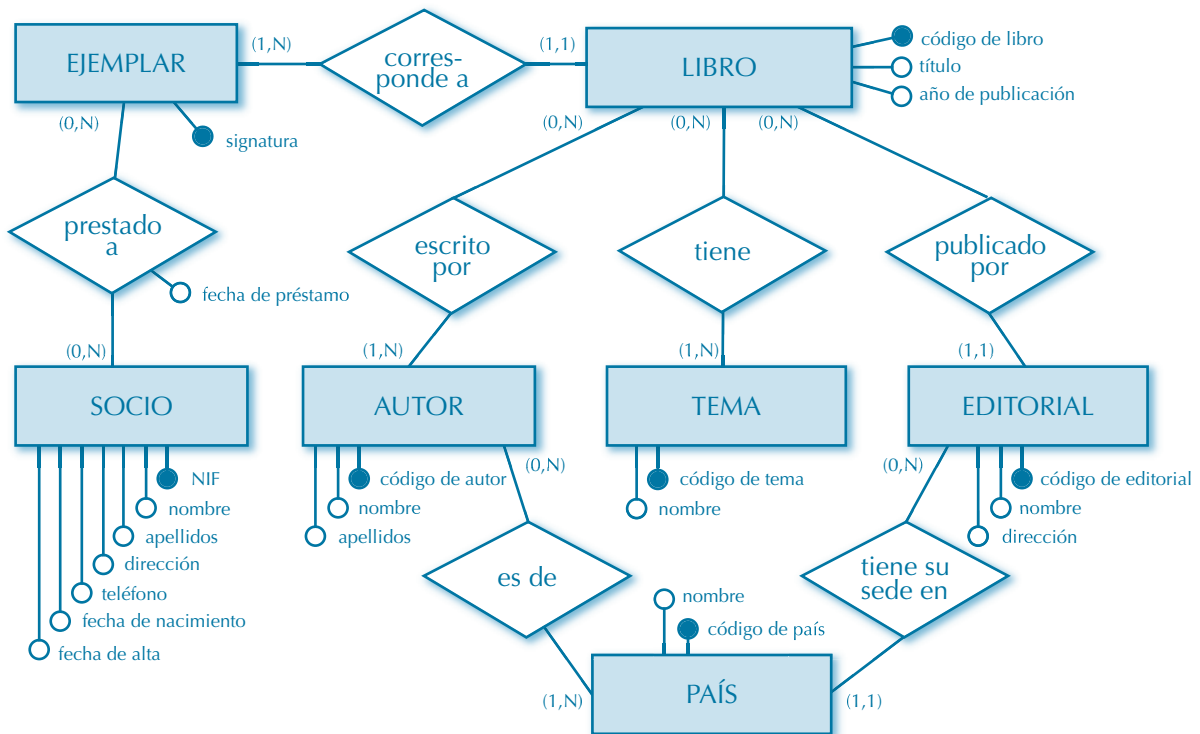


Figura 2.21. Ejemplo de diagrama entidad/relación

Se ha dotado de clave primaria a todas las entidades. Las cardinalidades de cada relación responden a los requerimientos especificados, pero intentan incluir las mínimas restricciones posibles. Por ejemplo, la cardinalidad de “Autor” hacia “Libro” es de 0:N, ya que un autor podría serlo de más de un libro (de ahí la N), pero en ningún momento se especifica que todas las ocurrencias de “Autor” deban corresponder obligatoriamente a autores de libros de esta biblioteca (quizás la base de datos resultante arranque con una carga inicial de una serie de autores, independientemente de que la biblioteca cuente con alguna de sus obras). Por eso figura cero como modalidad mínima.

PARA SABER MÁS



Un error habitual a la hora de representar un diagrama entidad/relación es identificar como entidad la organización cuya problemática hay que informatizar (y que, en muchos casos, corresponde con el cliente que encargó la aplicación informática asociada a la base de datos). Ejemplos de elementos que no deben representarse como entidades, quedando al margen del diagrama, serían el centro de enseñanza de las figuras 2.7 y 2.8, la empresa de la figura 2.10, la consultora financiera de la 2.12, el concesionario de automóviles de la 2.13 o la biblioteca de la 2.21.



Actividad propuesta 2.7

Representar el diagrama entidad/relación para el siguiente supuesto práctico:

Se desea almacenar información sobre la red de metro, sabiendo que los trenes están asignados a una línea determinada, cada uno se guarda en una cochera determinada (que está en una estación concreta) y en cada línea cada estación tiene un número de orden.



Actividad propuesta 2.8

Representar el diagrama entidad/relación para las entidades, relaciones y modalidades obtenidas en las actividades propuestas 2.1 a 2.4, añadiendo los atributos que se considere necesarios.

2.3. Diagrama entidad/relación extendido

Tal cual fue concebido al principio el diagrama entidad/relación no daba solución a todos los problemas que planteaba la modelización conceptual de datos, motivo por el que algunos autores le añadieron restricciones que, en algunos casos, ampliaban su magnitud semántica. Aunque no hay unanimidad sobre la necesidad e idoneidad de dichos añadidos ni sobre la notación a utilizar, estos son algunos de los elementos más reconocibles del diagrama entidad/relación extendido:

- *Exclusividad.* La existencia de una relación impide la existencia de otra. Supongamos un centro educativo donde un profesor puede dar clase o bien dedicarse a un trabajo de investigación, pero no hacer las dos cosas a la vez:

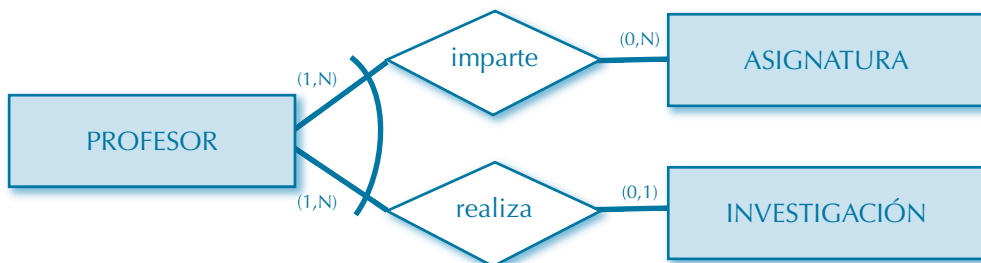


Figura 2.22. Exclusividad

- *Jerarquías*. Se relacionan las ocurrencias de una entidad (supertipo) con las de varias entidades (subtipos) que, además de compartir los atributos del supertipo, cuentan con atributos propios. En el siguiente ejemplo asociado al sistema de información de una agencia inmobiliaria se muestra el supertipo “Alquiler” y sus subtipos derivados, cada uno de ellos con sus atributos:

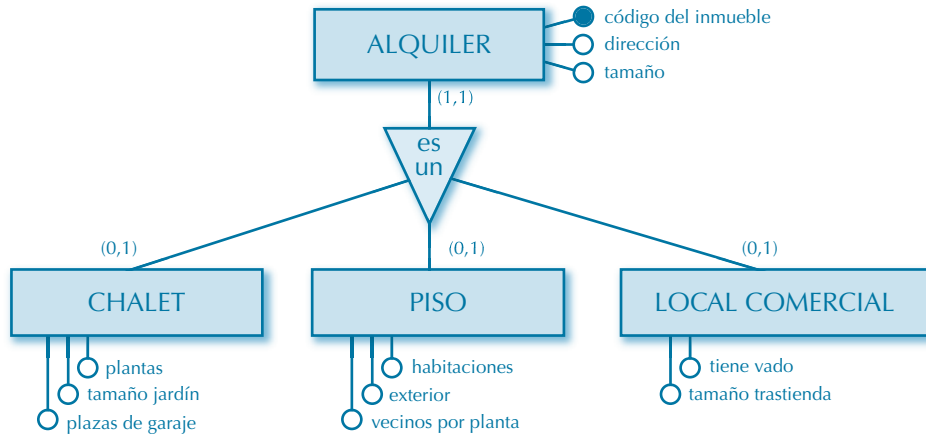


Figura 2.23. Jerarquía

Cada ocurrencia de “Chalet”, “Piso” y “Local comercial” tiene una ocurrencia correspondiente en “Alquiler”. Como todas las ocurrencias de los subtipos también lo son del supertipo, la modalidad es invariable (0..1 en el lado de los subtipos y 1..1 en el del supertipo).

Si la idea de jerarquía parte del supertipo, hablamos de *especialización* (ciertos atributos tienen sentido para algunas ocurrencias, pero no para otras, por lo que se decide crear entidades dependientes de la original). Si, en cambio, la idea parte de los subtipos, hablamos de *generalización* (entidades aisladas comparten varios atributos, motivo por el que se crea una entidad que los agrupe).

- *Agregación*. Ocurrencias de varias entidades constituyen las partes de un todo representado en una ocurrencia de otra entidad. Vemos un ejemplo en el que la agregación de varios componentes conforma un automóvil:

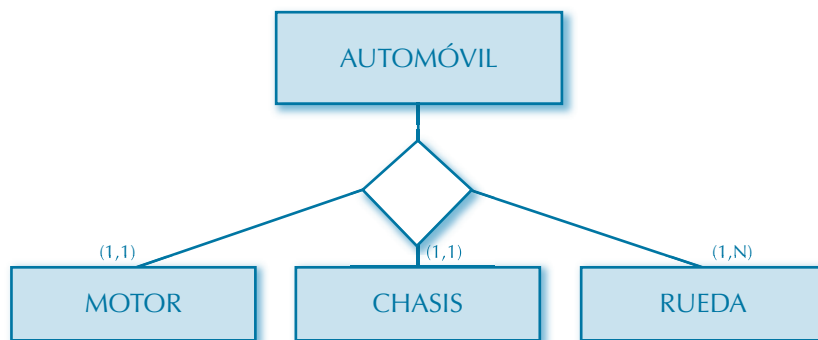


Figura 2.24. Agregación

PARA SABER MÁS



Existen diversas herramientas gráficas para diseñar diagramas entidad/relación, tanto propietarias como pertenecientes al mundo del *software* libre. En algunos casos dichas aplicaciones forman parte de herramientas CASE.



Actividad propuesta 2.9

Buscar en Internet herramientas de ayuda a la representación gráfica de diagramas entidad/relación.

Resumen

Todo desarrollo de *software* parte de un análisis estructurado, uno de cuyos elementos clave es la descripción de los datos. Si bien el diccionario de datos ofrecerá información sobre el contenido de la base de datos resultante, en un contexto relacional el diagrama entidad/relación será el primer paso de diseño de dicha base de datos.

El diagrama entidad/relación categoriza el universo del discurso que se va a informatizar como un conjunto de entidades que cubren las distintas apariciones (ocurrencias) de objetos concretos o abstractos del mundo real y las relaciones entre dichas entidades, cuyas restricciones vienen definidas por los conceptos de cardinalidad y modalidad. Tanto entidades como relaciones cuentan con atributos que incluyen cada uno de los datos que se van a tratar.

De cara a ampliar la funcionalidad del diagrama entidad/relación, diversos autores concibieron el diagrama entidad/relación extendido, que incluye los conceptos de exclusividad, jerarquía y agregación de información.

EJERCICIOS PROPUESTOS

1. Representar un diagrama entidad/relación para el siguiente supuesto:
 - Los clientes de nuestro videoclub pueden alquilar varias películas, y cada película se puede alquilar a varios clientes.
 2. Representar un diagrama entidad/relación para el siguiente supuesto:
 - Cada división de nuestra empresa está compuesta de varios departamentos. Cada departamento tiene muchos empleados asignados, pero cada empleado solo trabaja para un departamento. Cada departamento está dirigido por un empleado, y ningún empleado puede dirigir más de un departamento.
 3. Ampliar el ejercicio anterior con el siguiente requisito:
 - Cada empleado tiene a otro empleado como jefe.
 4. Representar un diagrama entidad/relación para el siguiente supuesto:
 - Una ferretería vende distintos productos, y cada uno de ellos puede ser creado por varios fabricantes e incluso variar el precio dependiendo de su fabricante. Cada vez que un cliente compra uno o varios productos se genera una factura que incluye la fecha, los productos comprados, la cantidad de cada producto, el precio unitario y el porcentaje de impuestos sobre el total de la factura.
 5. Modificar el diagrama entidad/relación de la figura 2.21 para que cumpla con los siguientes requisitos:
 - Hay que almacenar el ISBN de cada libro.
 - Cada libro solo corresponderá a un tema concreto.
 - En vez de almacenar el país donde se encuentra la sede de cada editorial, se almacenarán todos los países en los que cada editorial tenga oficinas, así como la dirección de dichas oficinas.
-

ACTIVIDADES DE AUTOEVALUACIÓN

1. El modelo conceptual de datos representa la información del sistema:
 - ☐ a) En el momento de la carga inicial.
 - ☐ b) En un instante concreto a lo largo de la vida del sistema.
 - ☐ c) En el estado final al que pueden llegar los datos.
2. En el diagrama entidad/relación, una entidad:
 - ☐ a) Puede tener distintos atributos para cada ocurrencia.
 - ☐ b) Puede tener una clave primaria compuesta de varios atributos.
 - ☐ c) Puede tener dos ocurrencias con los mismos valores en todos sus atributos.
3. Varias entidades pueden asociarse mediante una relación n -aria:
 - ☐ a) Si sus ocurrencias están inmersas en el mismo proceso.
 - ☐ b) Si la acción representada por la relación ocurre simultáneamente en ocurrencias de todas las entidades.
 - ☐ c) Si la acción representada por la relación afecta a ocurrencias de todas las entidades.
4. Si una ocurrencia de una entidad no tiene por qué relacionarse con ocurrencias de otra entidad con la que existe una relación:
 - ☐ a) La modalidad mínima será cero en el lado de la segunda entidad.
 - ☐ b) La modalidad mínima será cero en el lado de la primera entidad.
 - ☐ c) La modalidad mínima será cero en ambos lados.
5. Si las ocurrencias de una entidad se identifican unívocamente mediante los valores de varios atributos:
 - ☐ a) Dichos atributos formarán la clave primaria en un orden concreto.
 - ☐ b) Dichos atributos formarán la clave primaria sin importar el orden en que estén definidos.
 - ☐ c) Dichos atributos deben agruparse en un solo atributo.

SOLUCIONES:

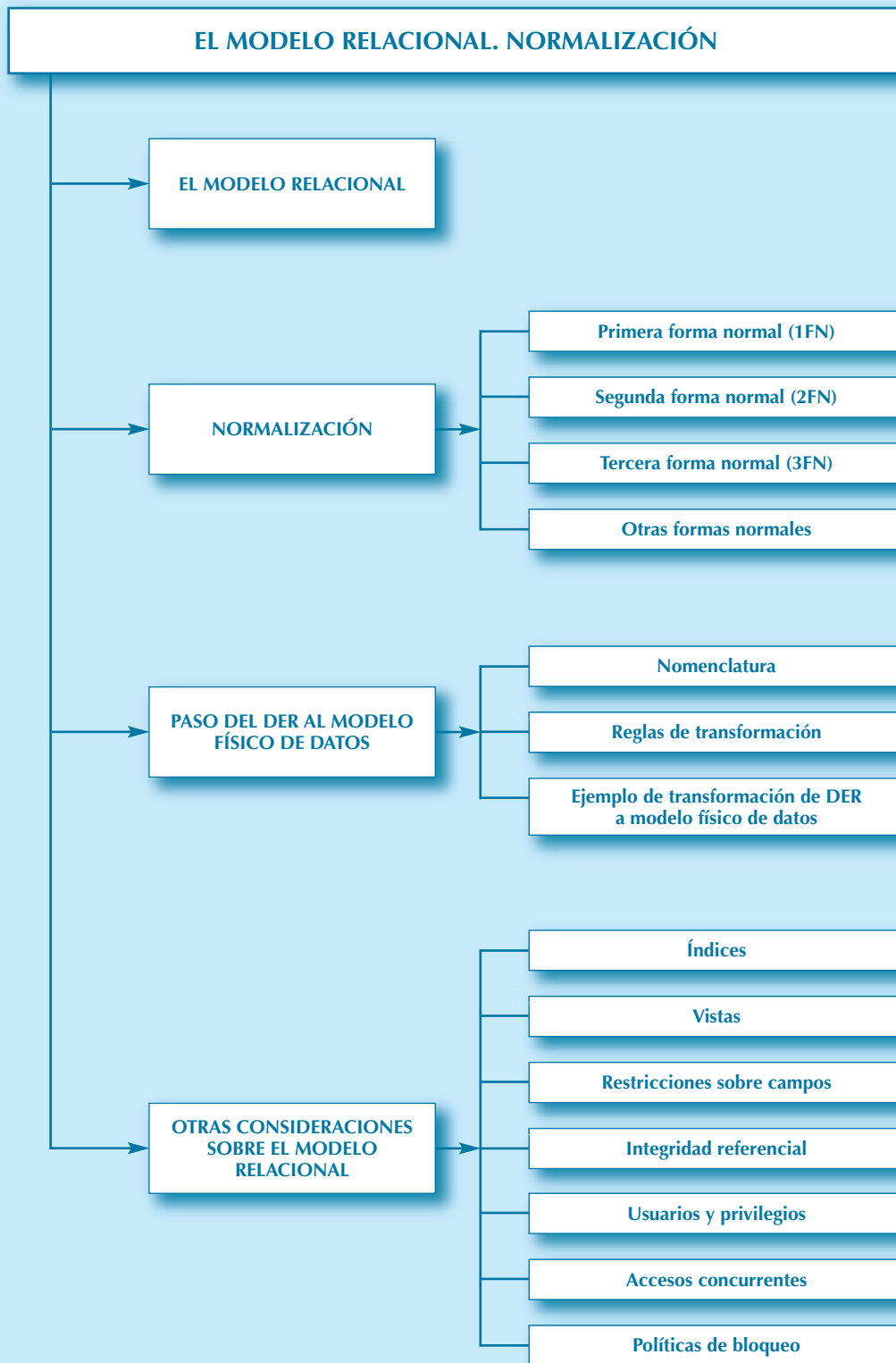
1. ☐ a ☐ b ☒ c2. ☐ a ☒ b ☐ c3. ☐ a ☒ b ☐ c4. ☒ a ☐ b ☐ c5. ☒ a ☐ b ☐ c

El modelo relacional. Normalización

Objetivos

1. Conocer los elementos básicos del modelo relacional.
2. Normalizar cualquier relación hasta su tercera forma normal.
3. Convertir un diagrama entidad/relación en un modelo físico de datos.
4. Comprender la utilidad y características de los índices en una base de datos relacional.
5. Familiarizarse con el concepto de vista.
6. Plantearse problemáticas de seguridad relacionadas con restricciones, integridad referencial, definición de usuarios y perfiles, accesos concurrentes y políticas de bloqueo.

Mapa conceptual del capítulo



Glosario

Clave ajena. En una base de datos relacional, presencia en una tabla de un campo que ejerce de clave primaria en otra tabla distinta.

Normalización. Técnica para eliminar redundancias en una base de datos relacional.

Perfil. Patrón de características comunes a varios usuarios de una base de datos.

Tabla. Estructura bidimensional de almacenamiento de información presente en una base de datos relacional.

Tipo de datos. Definición de los posibles valores que puede tomar un campo de una tabla relacional.

Variable. Datos almacenados en memoria temporal cuyo contenido puede cambiar a lo largo de la ejecución de un programa.

3.1. El modelo relacional

Definido por Edgar F. Codd en el IBM Research Center de San José (California) en 1970, el modelo relacional buscaba una solución a los problemas derivados de la rigidez estructural de las bases de datos jerárquicas y en red imperantes en la época. Presentado en sólidos términos matemáticos, independiza los datos de su tratamiento.

El elemento básico del modelo relacional es la *relación*, una estructura bidimensional que representa las entidades y algunas relaciones del diagrama entidad-relación. Las filas de una relación, llamadas *tuplas*, corresponden a las ocurrencias; cada tupla cuenta con una serie de atributos, cada uno de ellos con un valor. Vemos como ejemplo la relación “Socio”:

NIF	Nombre	Apellidos	Teléfono	Fecha de nacimiento	Fecha de alta
00789521T	Paula	Sanz González	619554687	13/09/1983	20/09/2012
09653801B	José Luis	García Viñals	667859621	05/02/1963	14/09/2011
38546958X	Javier	Peinado Martín	666932541	24/10/1978	05/05/2009
50687452Y	Ruth	Lázaro Cardenal	689330247	15/05/1981	04/09/2013

Figura 3.1. Relación

Llamamos *dominio* de un atributo al conjunto de valores que puede tomar para una ocurrencia concreta. Aunque de acuerdo a la definición teórica del modelo relacional cada dominio corresponde exclusivamente a los valores posibles (es decir, el dominio del atributo “Apellidos” es el conjunto de apellidos de los socios), la idea de dominio típicamente se asocia con la de *tipo de datos* (es decir, el conjunto de combinaciones de datos que podría constituir un dominio, de modo que desde un punto de vista técnico cualquier combinación de caracteres alfanuméricos, aunque no tenga sentido, podría representar los apellidos de un socio). Se identifican los siguientes tipos de datos base:

- Números enteros.
- Números decimales.
- Cadenas de caracteres.
- Fechas y horas.
- Valores lógicos (verdadero o falso).
- Objetos (ficheros binarios).

También se pueden definir conjuntos de valores a medida. Un ejemplo sería un atributo “día de la semana”, cuyos posibles valores fueran los contenidos en el conjunto {“lunes”, “martes”, “miércoles”, “jueves”, “viernes”, “sábado”, “domingo”}.

Hay un valor especial que constituye un tipo por sí mismo: el *valor nulo* o *NULL*, que representa ausencia de valor. No se debe confundir con el valor vacío (“” en un campo de tipo cadena de caracteres, o 0 en un campo numérico).

PARA SABER MÁS



El inglés Edgar Frank “Ted” Codd está reconocido de forma unánime como el padre de las bases de datos relacionales. Preocupado por la tendencia de los fabricantes de SGBDR a obviar elementos fundamentales de su propuesta de modelo relacional, a mediados de los años ochenta aportó una serie de definiciones conocidas como “las 12 reglas de Codd” (realmente son trece, numeradas del cero al doce) que todo SGBDR debe cumplir. A pesar de que algunas de ellas son muy complejas de seguir, la mera existencia de las 12 reglas supuso una declaración de intenciones sobre el camino que debía tomar el mundo de las bases de datos.

3.2. Normalización

También a Codd se debe la definición de una serie de normas cuya aplicación elimina las redundancias de información en una solución relacional. La técnica es conocida como normalización, y consiste en llevar todas las relaciones a determinados estados llamados formas normales. A continuación se describen dichas formas normales y cómo se llega hasta ellas.

3.2.1. Primera forma normal (1FN)

Una relación está en primera forma normal (1FN) si todos sus valores son atómicos, es decir, cada valor de los dominios de todos los atributos es único. En el siguiente ejemplo (clave primaria en negrita) vemos un atributo, “Teléfono”, cuyos valores no son atómicos, sino repetitivos:

NIF	Nombre	Apellidos	Teléfono
00789521T	Paula	Sanz González	619554687 915196347
09653801B	José Luis	García Viñals	667859621 914079880 913200121
50687452Y	Ruth	Lázaro Cardenal	689330247

Figura 3.2. Relación sin normalizar

Una primera solución para alcanzar la 1FN consiste en atomizar el atributo “Teléfono”, del siguiente modo:

NIF	Nombre	Apellidos	Teléfono
00789521T	Paula	Sanz González	619554687
00789521T	Paula	Sanz González	915196347
09653801B	José Luis	García Viñals	667859621
09653801B	José Luis	García Viñals	914079880
09653801B	José Luis	García Viñals	913200121
50687452Y	Ruth	Lázaro Cardenal	689330247

Figura 3.3. Relación en 1FN

Esta solución implica una fuerte redundancia (“Nombre” y “Apellidos” se repiten por cada teléfono), e invalida a “NIF” como clave primaria, obligando a ampliar dicha clave primaria con el atributo “Teléfono”. Por ese motivo se propone una solución más elaborada consistente en dividir la relación original en dos (una con las personas y otra con los teléfonos), vinculándolas mediante los valores de la clave primaria original:

NIF	Nombre	Apellidos
00789521T	Paula	Sanz González
09653801B	José Luis	García Viñals
50687452Y	Ruth	Lázaro Cardenal

NIF	Teléfono
00789521T	619554687
00789521T	915196347
09653801B	667859621
09653801B	914079880
09653801B	913200121
50687452Y	689330247

Figura 3.4. Relaciones en 1FN

La 1FN es parte de la definición del modelo relacional, por lo que su cumplimiento es obligatorio.

3.2.2. Segunda forma normal (2FN)

Una relación está en segunda forma normal (2FN) si cumple las siguientes reglas:

- Está en 1FN.
- Todos los atributos que no forman parte de la clave primaria dependen de ella por completo.

La relación siguiente ilustra el *stock* de una librería. La clave primaria está compuesta por dos atributos (“Código de libro” y “Código de tienda”), pero el atributo “Dirección” no depende de toda la clave, sino únicamente del atributo “Código de tienda”. Por ese motivo se repite la dirección de la tienda 9, con la consiguiente redundancia de información:

Cód. libro	Cód. tienda	Cantidad	Dirección
342	12	3	C/Luchana, 34
268	9	1	Pº de la Castellana, 132
250	10	5	C/General Ricardos, 145
181	9	7	Pº de la Castellana, 132

Figura 3.5. Relación en 1FN, pero no en 2FN

En este caso, el proceso de normalización obliga a dividir la relación en dos, una con la información de la tienda y otra con la del *stock*:

Cód. libro	Cód. tienda	Cantidad
342	12	3
268	9	1
250	10	5
181	9	7

Cód. tienda	Dirección
12	C/Luchana, 34
9	Pº de la Castellana, 132
10	C/General Ricardos, 145

Figura 3.6. Relaciones en 2FN

Nótese que la 2FN solo se puede violar si la clave primaria está compuesta por más de un atributo, por lo que toda relación en 1FN cuya clave primaria esté formada por un solo atributo también está en 2FN.

Actividad propuesta 3.1

Explicar por qué la siguiente relación no está en 2FN y convertirla en dos relaciones en 2FN:

Cód. factura	CIF cliente	Razón social	Fecha factura	Importe factura
0032	A280029013	Gráficas Martín, S.A.	04/01/2014	2.045,34 €
0044	B289388922	Díaz Distribuciones	08/12/2013	1.340,80 €
0102	A280211293	Transportes García	24/07/2014	580,00 €
0056	B289388922	Díaz Distribuciones	18/03/2014	1.014,03 €



3.2.3. Tercera forma normal (3FN)

Una relación está en tercera forma normal (3FN) si cumple las siguientes reglas:

- Está en 2FN.
- Todos los atributos que no forman parte de la clave primaria son independientes entre sí, es decir, no dan información sobre otros atributos de la relación.

El siguiente ejemplo ilustra una relación con información sobre empleados. Todos los atributos dependen directamente de la clave primaria (“Código de empleado”) excepto “Nombre de departamento”, que depende de “Código de departamento”:

Cód. emp.	Nombre	Apellidos	Dirección	Cód. dpto.	Nombre dpto.	Fecha nac.
12	Paula	Sanz González	C/Mtnéz. Izqdo., 40	3	Financiero	13/09/1983
268	José Luis	García Viñals	Pº Melancólicos, 1	2	Informática	05/02/1963
250	Javier	Peinado Martín	C/Guitarra, 7	5	RRHH	24/10/1978
181	Ruth	Lázaro Cardenal	C/Torrelaguna, 64	3	Financiero	15/05/1981

Figura 3.7. Relación en 2FN, pero no en 3FN

La información sobre el departamento constituirá una nueva relación:

Cód. emp.	Nombre	Apellidos	Dirección	Cód. dpto.	Fecha nac.
12	Paula	Sanz González	C/Mtnéz. Izqdo., 40	3	13/09/1983
268	José Luis	García Viñals	Pº Melancólicos, 1	2	05/02/1963
250	Javier	Peinado Martín	C/Guitarra, 7	5	24/10/1978
181	Ruth	Lázaro Cardenal	C/Torrelaguna, 64	3	15/05/1981

Cód. dpto.	Nombre dpto.
3	Financiero
2	Informática
5	RRHH

Figura 3.8. Relaciones en 3FN

Actividad propuesta 3.2



Se desea almacenar información sobre citas médicas, sabiendo que cada doctor puede ver a varios pacientes y cada paciente puede pedir cita con doctores distintos. Organizar los siguientes atributos en relaciones normalizadas hasta la 3FN:

- NIF de doctor
- Nombre de doctor
- Dirección de doctor
- NIF de paciente
- Nombre de paciente
- Teléfono de paciente
- Fecha de la cita

3.2.4. Otras formas normales

Inicialmente Codd definió las reglas relativas a las tres primeras formas normales, que todo diseño relacional que se precie debe cumplir. No obstante más adelante se amplió la teoría de la normalización con la forma normal de Boyce-Codd (FNBC), cuarta forma normal (4FN), quinta forma normal (5FN), forma normal de dominio/clave (DKNF) y sexta forma normal (6FN). De enorme contenido teórico, fuerzan restricciones que hacen su aplicación poco recomendable en muchos entornos reales.

3.3. Paso del DER al modelo físico de datos

El objetivo del diagrama entidad-relación y las técnicas de normalización es proporcionar el mejor diseño posible para una futura base de datos. En una implementación relacional la información se almacena en forma de *tablas* con *campos* y *registros*, a modo de las relaciones, los atributos y las tuplas del modelo relacional (coincidentes, a su vez, con las entidades y relaciones, los atributos y las ocurrencias del diagrama entidad-relación).

CUADRO 3.1. Comparación entre elementos del paradigma relacional

Diagrama entidad-relación	entidad o relación	ocurrencia	atributo
Modelo relacional	relación	tupla	atributo
Modelo físico de datos	tabla	registro	campo

Se muestra la tabla TSocio, resultante de la relación “Socio” vista anteriormente:

valor

campo

TSocio

cNIF	cNombre	cApellidos	cTelefono	dNacimiento	dAlta
00789521T	Paula	Sanz González	619554687	13/09/1983	20/09/2012
09653801B	José Luis	García Viñals	667859621	05/02/1963	14/09/2011
38546958X	Javier	Peinado Martín	666932541	24/10/1978	05/05/2009
50687452Y	Ruth	Lázaro Cardenal	689330247	15/05/1981	04/09/2013

registro

Figura 3.9. Tabla

3.3.1. Nomenclatura

A partir de este momento vamos a definir una nomenclatura común a todos los elementos de modelo físico y base de datos referidos. Para ello se usará como punto de partida la notación húngara, definida en los años 70 por Charles Simonyi, programador húngaro de Xerox. Simonyi estableció unas reglas de nominación de variables que aportaba información sobre su ámbito y tipo de datos. Centrada originalmente en código fuente de programación, se presenta a continuación una adaptación reducida de la notación húngara a los nombres de elementos de una base de datos relacional, según los siguientes criterios:

- *Nombres de tablas.* Se prefijarán con una “T” mayúscula (TEmpleado, TCliente).
- *Nombres de campo.* El prefijo, una letra minúscula, indicará el tipo de datos del campo, de acuerdo a los siguientes tipos básicos:
 - Números: “n” (nCantidad, nTotal), de *number*.
 - Cadenas de caracteres: “c” (cNombre, cCiudad), de *character*.
 - Fechas: “d” (dNacimiento, dAlta), de *date*. Nótese que ya no es necesario especificar la palabra “fecha” (dFechaNacimiento sería redundante).
 - Valores lógicos: “l” (lSexo, lPensionista), de *logical*.
 - Objetos: “o” (oFoto, oDocumentoXML), de *object*.
- *Nombres (genérico).* Se evitará el uso de caracteres de alfabetos locales, como la ñ, la ç, las vocales acentuadas o con diéresis, etc. Únicamente se aceptarán como válidos los siguientes caracteres:
 - Letras del alfabeto inglés en mayúscula y minúscula.
 - Números.
 - Signo de subrayado o guion bajo (“_”).

El primer carácter de un nombre siempre será una letra.

- *Identificadores.* Todo campo creado para identificar de forma unívoca los registros de una tabla (código de cliente, código de libro, código de país) llevará como sufijo las letras “ID” (identificador) en mayúscula (nClienteID, nLibroID, nPaisID).

A pesar de que esta nomenclatura puede resultar insuficiente en el entorno de trabajo de un SGBD real, nos servirá como marco de referencia común para los ejemplos de este texto. Queda a discreción del lector ampliarla y/o adaptarla como convenga a sus necesidades.

PARA SABER MÁS



Tras su andadura en Xerox, Charles Simonyi desarrolló una exitosa carrera en Microsoft, donde supervisó el desarrollo de las herramientas Word y Excel a principios de los años ochenta. Con once patentes publicadas, se ha dedicado a la filantropía de forma activa y es el único turista espacial en repetir su aventura (2007 y 2009).



Figura 3.10. Charles Simonyi

3.3.2. Reglas de transformación

Se propone la siguiente representación gráfica de modelo físico de datos:

- Las tablas se representan como un rectángulo con el nombre de la tabla en la parte superior y la lista de campos en la inferior. Los campos que conformen la clave primaria irán subrayados y en orden:

TAutor
<u>nAutorID</u> cNombre cApellidos

Figura 3.11
Representación gráfica
de una tabla

- La modalidad irá implícita en la terminación de las líneas que relacionan tablas, del siguiente modo:

Figura 3.12. Modalidad 0..1



Figura 3.13. Modalidad 1..1

Figura 3.14. Modalidad 1..N



Figura 3.15. Modalidad 0..N

PARA SABER MÁS



Debido al aspecto de la línea que representa la modalidad 1..N, esta notación del modelo físico de datos es conocida como “de pata de gallo” (*crow's foot*, literalmente en inglés “de pata de cuervo”).

En algunos casos, las relaciones generan tablas (se verá en breve). A la hora de dar nombre a dichas tablas se pueden seguir dos criterios:

- Utilizar un sustantivo que represente la acción implícita en la relación (“cliente contrata servicio” generaría una tabla TContrato, “usuario compra producto” generaría TCompra).
- Concatenar los nombres de las entidades relacionadas (“profesor imparte asignatura” generaría TProfesorAsignatura).

PARA SABER MÁS



La elección de nombres de tablas, campos, variables, etc., es una tarea crucial de cara a la correcta comprensión del diseño de un sistema de información. Se dice que un programador debe dedicar más tiempo a pensar qué nombre dar a una variable que el que otro programador tardará en descubrir para qué se emplea dicha variable.

La transformación de componentes del diagrama entidad-relación en elementos del modelo físico de datos sigue las siguientes reglas:

- Toda entidad se convierte en una tabla.
- Todo atributo pasa a ser un campo. Los atributos marcados como parte de la clave primaria se convierten en campos de la clave primaria de la nueva tabla, conservando su orden.
- Las relaciones presentan una casuística basada en su cardinalidad. Uno de los objetivos fundamentales a la hora de diseñar una base de datos es evitar la proliferación de valores nulos:
 - *Relaciones M:N, ternarias y n-arias.* Se convierten en tabla. Su clave primaria será la concatenación de las claves primarias de las entidades que relacionan.

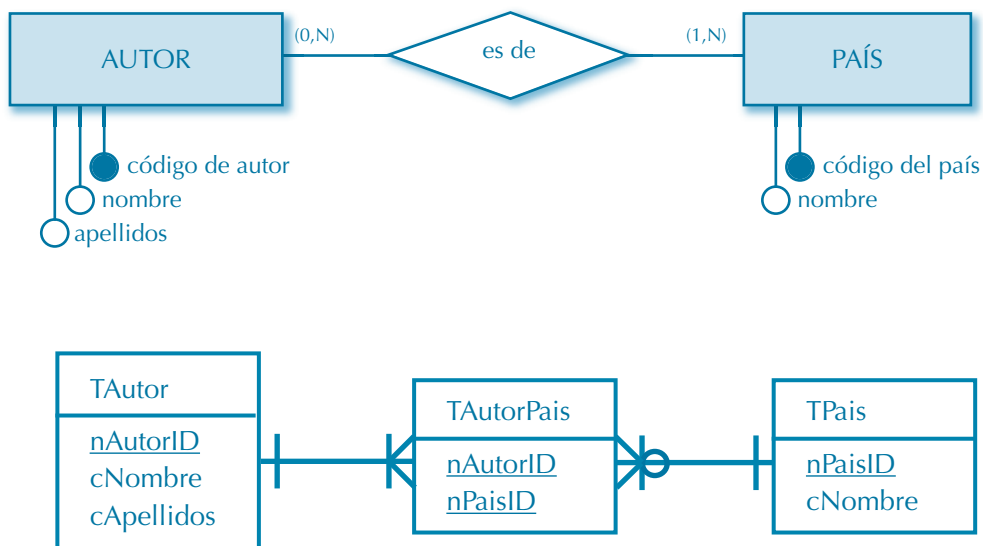


Figura 3.16. Relación M:N

- *Relaciones 1:N.* Se presentan tres posibilidades básicas:
 - Si la relación tiene algún atributo, se convierte en tabla:

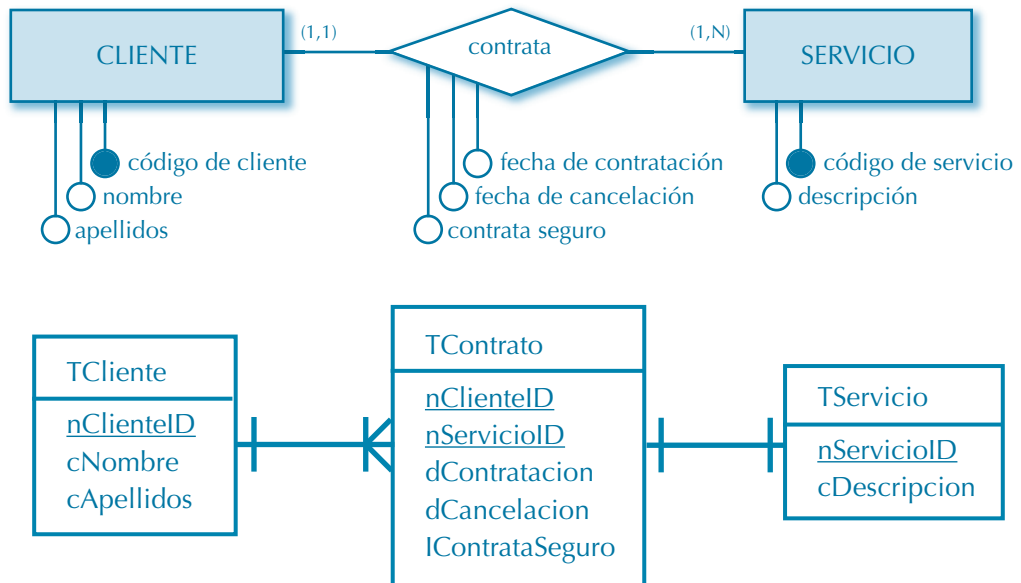


Figura 3.17. Relación 1:N

Es posible que se dé una circunstancia que haya que evaluar en el momento de la transformación: si algún atributo de la relación debe formar parte de la clave primaria o no. En el presente ejemplo, la relación cuenta con tres atributos, “Fecha de contratación”, “Fecha de cancelación” y “Contrata seguro”. Si la especificación de requisitos indica que un cliente puede contratar el mismo servicio más de una vez (previa cancelación), la clave primaria de TContrato (formada por la concatenación de las claves primarias de las entidades relacionadas) no garantizaría la unicidad (podría haber dos registros con el mismo nClienteID y nServicioID). En ese caso, se debería añadir la fecha de contratación a la clave (entendiendo que no puede haber un cliente que contrate el mismo servicio dos veces en la misma fecha):



Figura 3.18. Relación 1:N con atributos de relación en la clave

Este tipo de problemática tiene que ver con la dimensión temporal de la información almacenada en una base de datos, y suele requerir intervención especial por parte del analista, que evaluará cuál es la solución más eficiente para cada caso.

- Si la modalidad mínima en el lado de cardinalidad 1 es 1 y no hay atributos, la relación desaparece. Los atributos que conforman la clave primaria de la entidad de modalidad máxima 1 se propagarán a la entidad de modalidad máxima N:

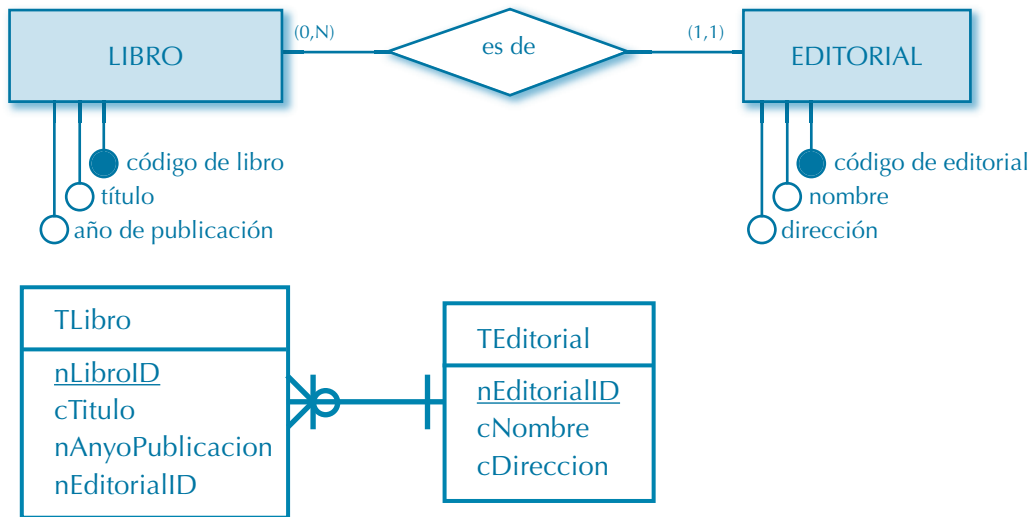


Figura 3.19. Relación 1:N con modalidad 1..1

Debemos referirnos aquí a un nuevo concepto. Cuando el campo o campos que conforman la clave primaria en una tabla aparecen en otra tabla distinta, se dice que conforman una *clave ajena* o *foránea* (*foreign key*). En el ejemplo nEditorialID es clave primaria en TEditorial y clave ajena en TLibro.

- Si la modalidad mínima en el lado de cardinalidad 1 es 0, la relación se convierte en tabla.

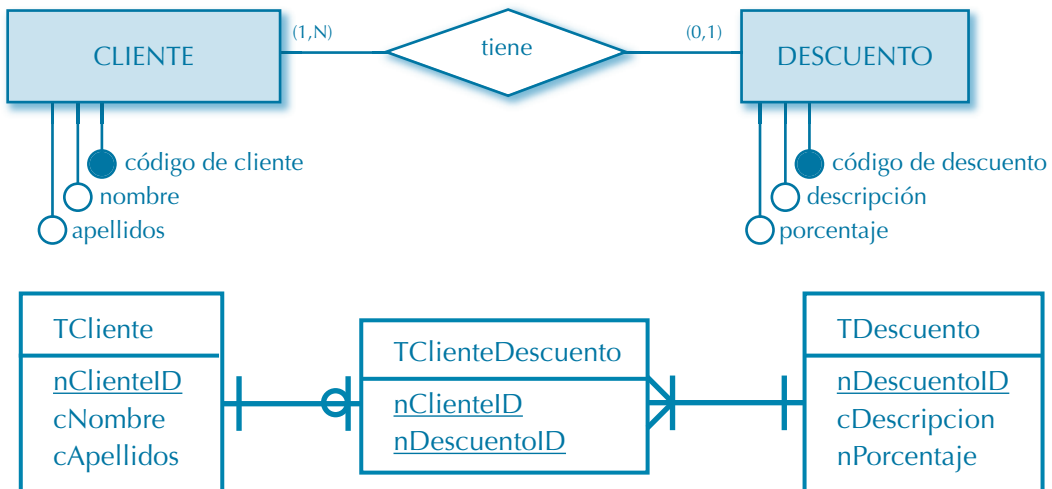


Figura 3.20. Relación 1:N con modalidad 0..1

Si, en el presente ejemplo, se hubiese optado por propagar `nDescuentoID` a `TCliente` se podrían generar valores nulos en dicho campo, ya que no todos los clientes tienen descuento.

- **Relaciones 1:1.** Es la casuística más compleja. El objetivo es evitar a toda costa que queden campos sin valor. Generalmente requiere un estudio detallado de las circunstancias concretas de cada caso, pero se pueden definir unas reglas para ciertas situaciones:
 - Si ambas modalidades son 0..1, la relación genera una tabla. Obsérvese el siguiente ejemplo, donde se relacionan empleados con ubicaciones físicas dentro de una oficina, dándose la posibilidad de que haya empleados sin ubicación (personal de limpieza o mensajería) y ubicaciones vacías (mesas sin ocupar). Propagar cualquiera de las claves primarias a la otra entidad ocasionaría valores ausentes en campos (empleados sin valor en su código de ubicación o ubicaciones sin valor en su código de empleado):

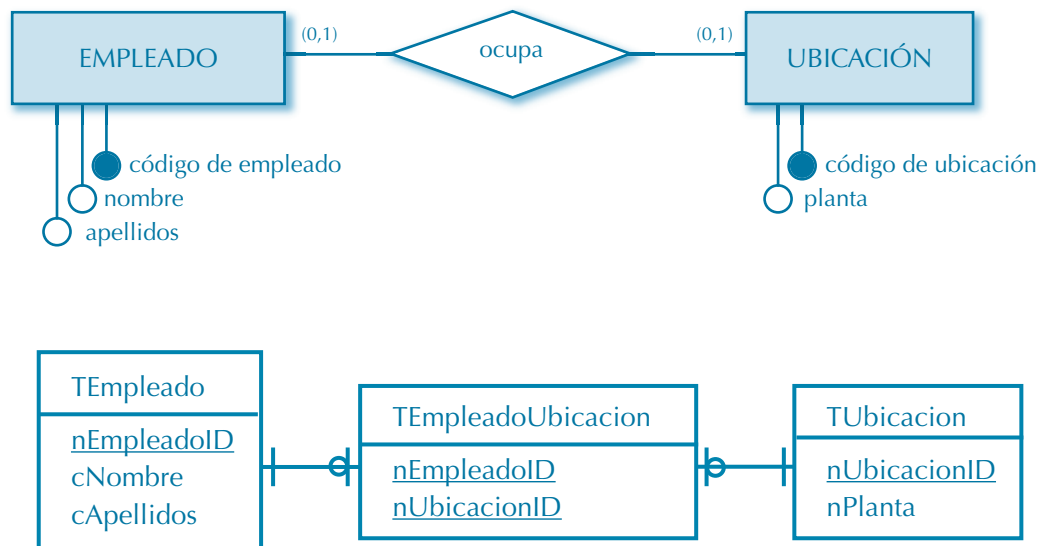


Figura 3.21. Relación 1:1 con modalidades 0..1

- Si una modalidad es 0..1 y la otra 1..1, la clave de la entidad con modalidad mínima 1 se propaga a la entidad con modalidad mínima 0. De forma más intuitiva, en el siguiente ejemplo se puede ver que todo chalet es un inmueble, pero no todo inmueble es un chalet, por lo que lo lógico es que la clave de “Inmueble” se propague a “Chalet”, y no al revés:

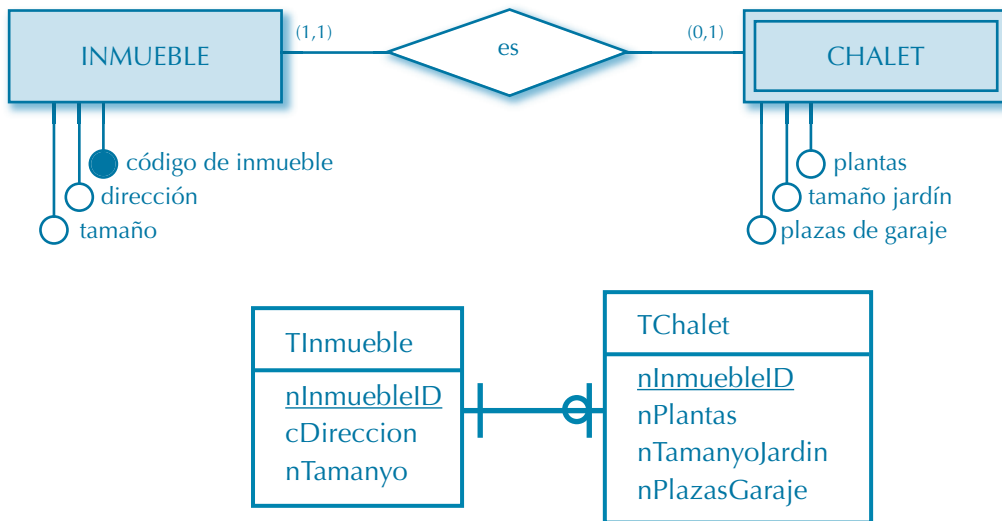


Figura 3.22. Relación 1:1 con modalidad 0..1

- Nótese que en este caso la clave primaria de TInmueble se propaga a TChalet como clave ajena, pero también funcionará en TChalet como clave primaria (se garantiza la unicidad), aunque se rellene con valores ya existentes en TInmueble. En este caso, en el que una entidad débil necesita de la clave primaria de la entidad fuerte de la que depende, se dice que hay una relación de *dependencia en identificación*. Cuando la entidad débil puede generar su propia clave primaria se habla de *dependencia en existencia*. Si, en vez de conformar la clave primaria de TChalet por completo, nInmuebleID fuese parte de una clave compuesta, la relación seguiría siendo de dependencia en identificación.
- Si ambas modalidades son 1..1, pero una de las entidades es débil, la clave de la entidad fuerte se propaga a la débil. Supongamos una relación 1 a 1 entre clientes y carteras de inversión donde la cartera de inversión se crea exclusivamente a partir del cliente:

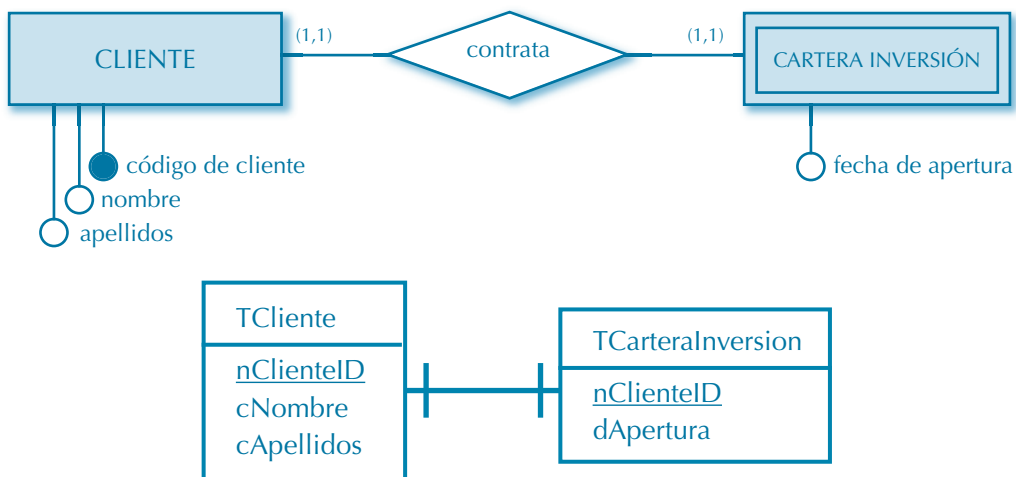


Figura 3.23. Relación 1:1 con entidad débil

- Si ambas modalidades son 1..1 y no hay entidades débiles, o bien si existe algún atributo de relación, habrá que estudiar en detalle cuál es la opción adecuada en cada caso, dependiendo de la semántica de los datos almacenados y de otros detalles técnicos, como el número estimado de accesos a cada una de las tablas o el orden de consulta de la información.



Actividad propuesta 3.3

Como se comentó en el capítulo anterior, las relaciones reflexivas funcionan igual que las binarias, excepto por el hecho de que se están relacionando ocurrencias de la misma entidad que juegan papeles distintos (recuérdese el ejemplo de la figura 2.9). Crear relaciones reflexivas con las siguientes modalidades y convertirlas a modelo físico de datos:

- a) 1..N a 1..N
- b) 1..1 a 0..N
- c) 1..1 a 1..1
- d) 0..1 a 0..1



Actividad propuesta 3.4

Las relaciones ternarias y n -arias generan tabla. Tomar el diagrama de la figura 2.16, añadir atributos a cada una de las cuatro entidades y representar su paso a modelo físico de datos generando la tabla de relación correspondiente.

- La transformación de elementos correspondientes al diagrama entidad-relación extendido suele conllevar pérdida de semántica, y exige en algunos casos la inclusión de mecanismos de control en la base de datos resultante. Por ese motivo, algunos autores no recomiendan su uso, argumentando que la sintaxis del DER original cubre todas las necesidades de un diseño conceptual de datos.

En cualquier caso, si es necesario transformar elementos del DER extendido se pueden aplicar las siguientes reglas dentro del propio diagrama entidad/relación:

- Las relaciones exclusivas se tratan como relaciones normales. El control de exclusividad se deberá implementar posteriormente como restricción de la base de datos.

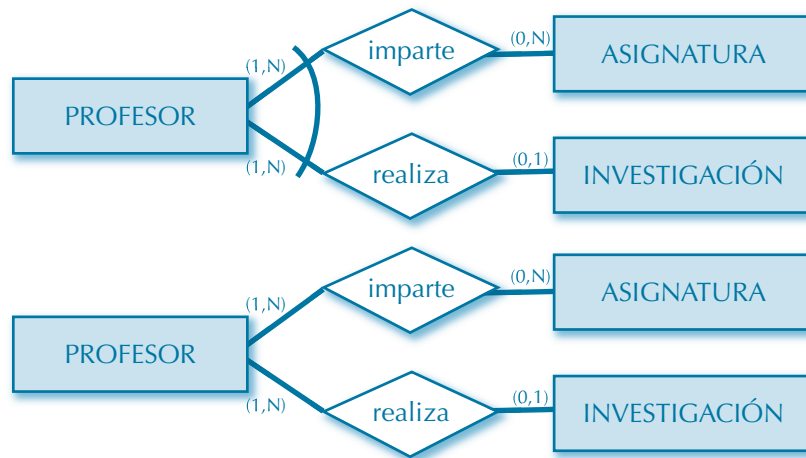


Figura 3.24. Tratamiento de relaciones exclusivas

- Las relaciones de jerarquía se convierten en relaciones 1:1 con modalidades 0..1 y 1..1.

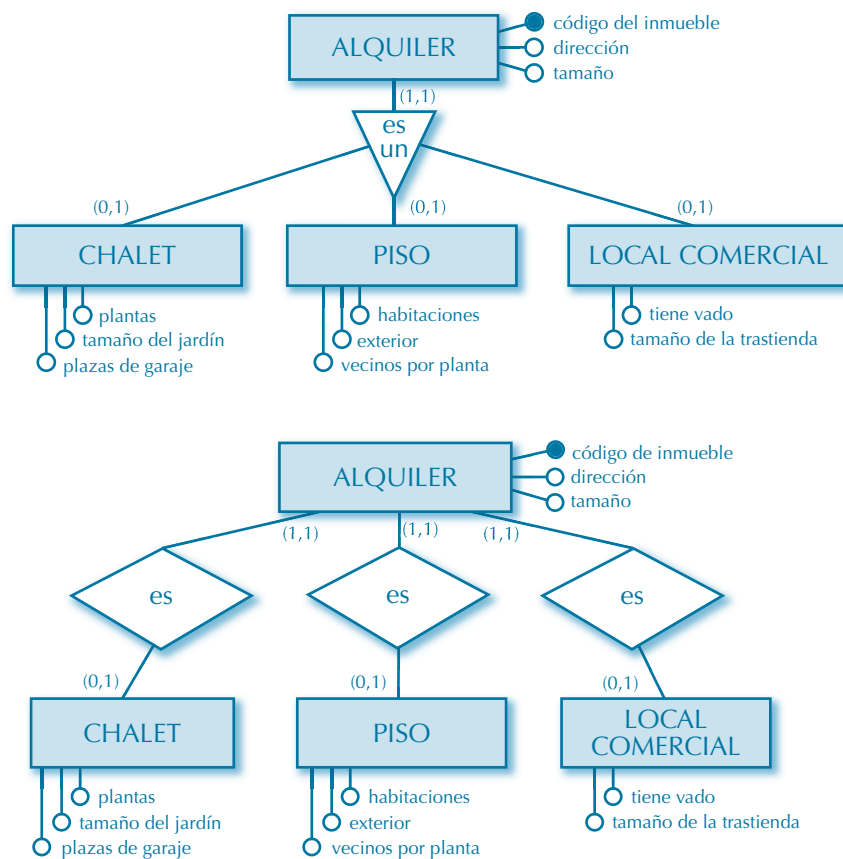


Figura 3.25. Tratamiento de relaciones jerárquicas

- Las relaciones de agregación se tratan como relaciones individuales entre cada entidad agregada y la entidad principal, en cuyo lado habría que determinar la modalidad de acuerdo a los requisitos especificados en cada caso.

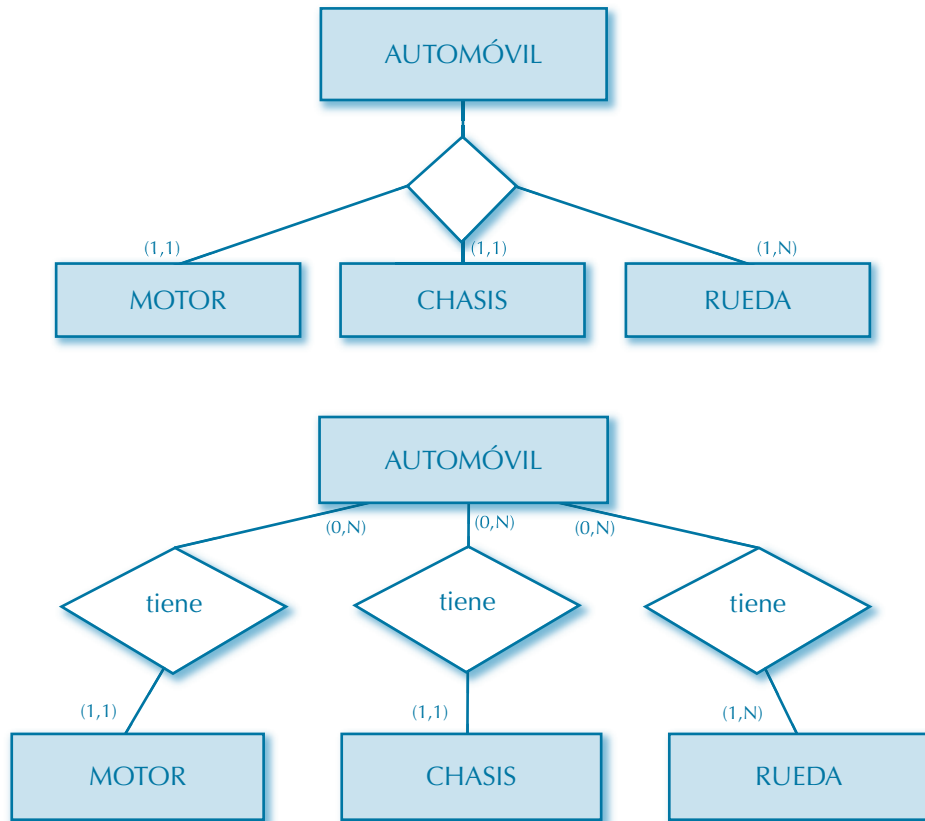


Figura 3.26. Tratamiento de relaciones de agregación

En este caso se ha asignado modalidad $0..N$ a las tres relaciones en el lado de “Automóvil”, suponiendo que “Motor” almacene modelos de motor, “Chasis” modelos de chasis y “Rueda” modelos de rueda, de modo que el mismo motor, el mismo chasis y la misma rueda pueden instalarse en varios automóviles. En otras circunstancias la modalidad en el lado de automóvil podría ser $1..N$, $1..1$ o incluso $0..1$.

3.3.3. Ejemplo de transformación de DER a modelo físico de datos

Tomemos como ejemplo de diagrama entidad-relación el presentado en el tema anterior:

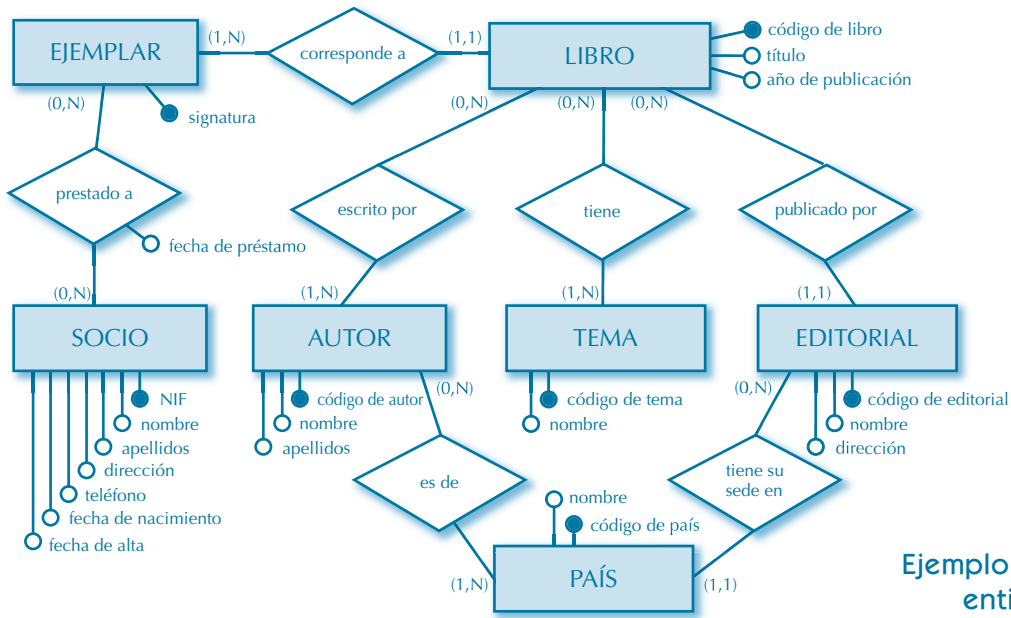


Figura 3.27
Ejemplo de diagrama entidad/relación

Esta sería su transformación a modelo físico de datos:

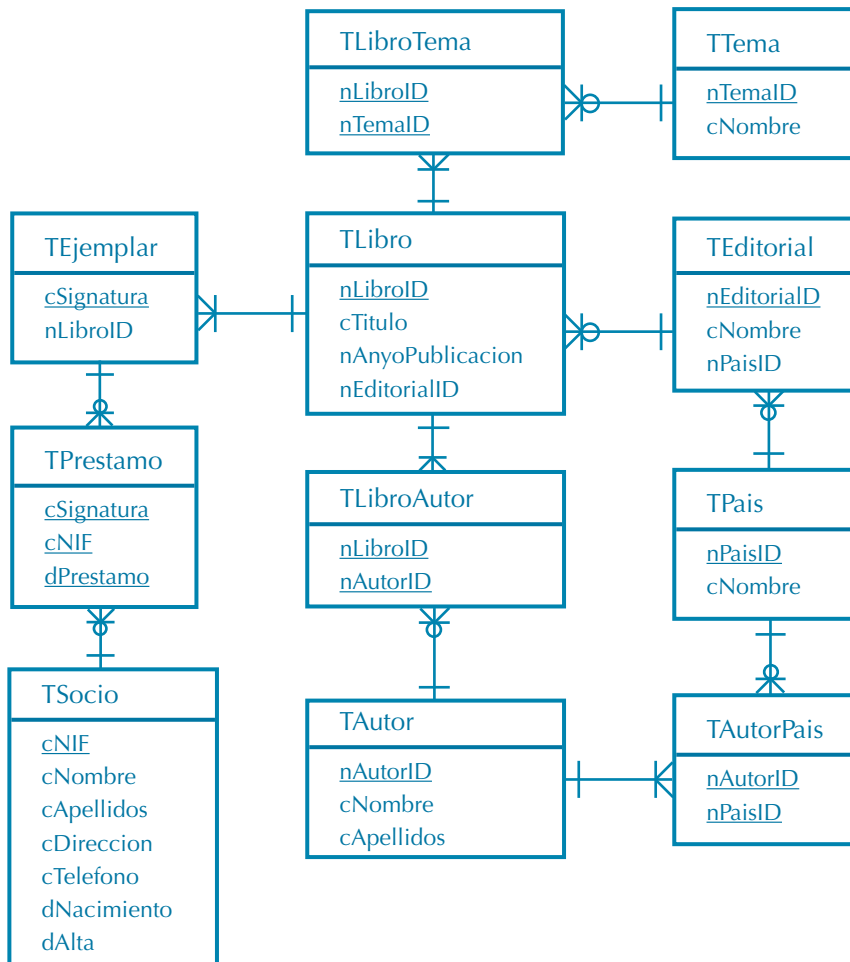


Figura 3.28
Ejemplo de modelo físico de datos



Actividad propuesta 3.5

Convertir a modelo físico de datos el resultado de las actividades propuestas 2.7 y 2.8 del capítulo anterior.

3.4. Otras consideraciones sobre el modelo relacional

3.4.1. Índices

El orden de presentación de la información en pantalla o listado no tiene por qué coincidir con su orden de almacenamiento, por lo que el modelo relacional provee herramientas que independizan ambos escenarios. De modo análogo a lo explicado en el tema 1 en relación con los ficheros indexados, las tablas de una base de datos relacional también cuentan con ficheros de índice asociados, fundamentales para buscar información y para acceder a ella de forma ordenada.

Como mínimo habrá un fichero de índice que relacione cada registro con su valor correspondiente de clave primaria. A partir de ahí, es tarea del diseñador definir los índices que se considere necesarios, si bien hay algunos criterios básicos:

- *Campos por los que se van a ordenar las consultas más habituales.* Miremos la tabla TSocio. Típicamente los bibliotecarios utilizarán una aplicación *software* mediante la que manipularán la información de la base de datos, y contarán con una pantalla donde poder dar de alta socios, modificar sus datos, eliminarlos o simplemente consultarlos. La clave primaria de TSocio es cNIF, pero ordenar a los socios por su NIF no aportará ventajas a los usuarios de la aplicación. Lo normal será que quieran ordenarlos por apellidos y nombre, o bien por fecha de alta. Para ello se deberían crear dos índices, uno por cApellidos + cNombre, y otro por dAlta. De esa forma, cada vez que un bibliotecario quiera consultar la lista de socios ordenados por apellido y nombre, el SGBD detectará que existe un índice por ambos campos, y leerá dicho índice en vez de recorrer la tabla secuencialmente.

Es necesario señalar la importancia del orden de campos en la definición de un índice (no es lo mismo ordenar por cApellidos + cNombre que por cNombre + cApellidos).

- *Campos accedidos con mucha frecuencia.* El ordenamiento de información que implican los índices no solo sirve para mostrar grupos de registros ordenados. Además permite buscar valores con mayor rapidez. Generalmente el bibliotecario buscará a los socios por sus apellidos, por lo que el índice cApellidos + cNombre también servirá para agilizar sus consultas.
- *Claves ajenas.* Es muy habitual ejecutar consultas que afectan a varias tablas relacionadas. Para localizar los libros publicados por una editorial tendríamos que seguir los siguientes pasos:
 1. Partiendo del nombre de la editorial (campo cNombre de TEditorial), ver cuál es el código de editorial de ese registro (campo nEditorialID de TEditorial).
 2. Buscar ese valor en la tabla de libros (TLibro).
 3. Obtener el valor correspondiente de cTitulo en TLibro.

Para una consulta como esta nos vendría muy bien que TLibro contase con un índice por nEditorialID. De lo contrario, la búsqueda tendrá que realizarse directamente sobre la tabla, cuyos datos pueden estar desordenados, o bien ordenados de acuerdo a un criterio que no satisfaga nuestra búsqueda.

A pesar de todas sus ventajas, la existencia de índices crea redundancia y ralentiza los procesos de inserción, actualización y borrado, al implicar la reordenación de los índices asociados. Por ese motivo es importante no abusar de su número ni de su tamaño (un índice compuesto por todos los campos de una tabla ocuparía más espacio que la propia tabla, y ralentizaría sustancialmente su acceso).

Actividad propuesta 3.6

Analizar la base de datos de la figura 3.28 y enumerar posibles índices para cada una de sus tablas, justificando la elección de acuerdo a su utilidad.



3.4.2. Vistas

Una vista es una consulta sobre una o varias tablas almacenada en la base de datos. De cara al usuario la información obtenida mediante una vista parece una tabla más, pero el uso de vistas incrementa el nivel de seguridad en el acceso a los datos y son más eficientes que las consultas efectuadas puntualmente, al encontrarse ya definidas en la base de datos.

No obstante las vistas no permiten recibir parámetros, es decir, no se pueden ejecutar para distintos valores de campos concretos de la base de datos. Una vista puede obtener una lista de clientes, o una relación de facturas cuyo importe esté dentro de un rango, pero no puede filtrar sus valores por un dato variable. Esa estaticidad provoca que muchos desarrolladores ignoren su existencia, especialmente si ya se han establecido otros mecanismos de seguridad.

3.4.3. Restricciones sobre campos

Además de dotarlo de un tipo de datos (elemento que se tratará en el tema siguiente), también se pueden definir las siguientes restricciones sobre un campo de una tabla:

- **UNIQUE.** Todos los valores del campo deben ser únicos, y no se permiten valores repetidos. Cuando una clave primaria está compuesta por un solo campo, dicho campo está obligado a cumplir la restricción UNIQUE, aunque esta no es exclusiva de los campos que conforman una clave primaria.
- **NOT NULL.** Prohíbe que un campo pueda tener valores nulos. Es una restricción obligatoria en los campos que conforman la clave primaria.
- **DEFAULT.** En caso de que no se especifique un valor concreto para un campo, en vez de dejarlo a NULL se le asigna un valor por defecto. El SGBD correspondiente ofrecerá herramientas para definir dicho valor.



Actividad propuesta 3.7

Decidir cuáles de las tres restricciones mencionadas en el apartado 3.4.3 deberían aplicarse a cada campo de cada tabla de la base de datos de la figura 3.28.

3.4.4. Integridad referencial

La existencia de tablas relacionadas mediante clave ajena ocasiona una problemática que atañe a la integridad de la información implicada. ¿Es posible borrar el registro correspondiente a un socio de la biblioteca que tiene préstamos registrados? ¿Se puede cambiar la clave primaria de una editorial relacionada con libros? Diseñar una base de datos también implica tomar decisiones de integridad referencial. Hay cuatro enfoques básicos:

- *Prohibir la operación.* Es la decisión más restrictiva. Impide el borrado o modificación de registros que tengan coincidencias por clave ajena en otra u otras tablas. Se puede implementar de modo que, sencillamente, la operación no se lleve a cabo, o bien que, además, genere un error.
- *Transmisión en cascada.* Se arrastra la modificación o el borrado a las tablas con registros relacionados (es decir, si se eliminase en TAuthor el registro correspondiente a un autor con libros también se eliminarían automáticamente los registros de TLibroAutor que vinculan TAuthor con TLibro).
- *Puesta a nulo.* Se permite la operación de modificación o borrado, pero poniendo a NULL los valores de las claves ajenas correspondientes (se podría borrar en TEditorial una editorial con libros, pero en los registros correspondientes de TLibro el campo nEditorialID quedaría a NULL). En este caso, las claves ajenas no pueden contar con la restricción NOT NULL.

Esta alternativa no es recomendable, ya que contradice una regla de eficiencia básica: evitar la proliferación de valores nulos.

- *Uso de valor por defecto.* Se permite la operación de modificación o borrado, pero asignando a las claves ajenas correspondientes su valor por defecto. Implica que ese valor esté definido.

3.4.5. Usuarios y privilegios

Los SGBD permiten definir una estructura de seguridad basada en los permisos que se concede a los usuarios. Para ello hay que tomar las siguientes decisiones:

- Asignar un *usuario* de la base de datos a cada usuario físico del SGBD o bien crear usuarios genéricos que puedan corresponder a varios usuarios físicos (administrador, operador, desarrollador, usuario no técnico, etc.).
- En el primero de los casos se pueden agrupar varios usuarios en *perfiles*, de modo que varias personas, cada una de ellas con su usuario de la base de datos, correspondan al perfil de administrador, al de desarrollador, etc.

- Para cada usuario o perfil hay que establecer qué tipo de acceso tiene al conjunto de objetos de la base de datos (bases de datos, tablas, tareas de administración, gestión de usuarios). Se puede definir un acceso de solo lectura a ciertos objetos, pero con la posibilidad de modificar otros, se puede impedir el acceso a partes de la base de datos, incluso dentro de tablas (si la empresa contrata a un becario para el Departamento de Recursos Humanos, tendrá que acceder a la tabla con los datos del personal de la empresa, pero no sería conveniente que pueda consultar el salario de los empleados).
- A veces un usuario de la base de datos no corresponde ni a un usuario físico ni a varios, sino a una aplicación *software* a través de la que el usuario físico interactuará con la base de datos de forma transparente.

A continuación se muestra un ejemplo de definición de perfiles y usuarios en un SGBD:

```

Perfil:      Admin
Usuarios:    Admin
Permisos:    Lectura, escritura, creación, borrado y backup sobre todas
              las bases de datos.
              Creación, modificación y borrado de perfiles y usuarios.
              Administración del servidor.

Perfil:      Operador
Usuarios:    Operador
Permisos:    Lectura, escritura, creación, borrado y backup sobre todas
              las bases de datos.
              Creación, modificación y borrado de perfiles y usuarios.

Perfil:      Desarrollo
Usuarios:    JimenezJ, GrandesA, GarciaF, PuigM, MachadoA
Permisos:    Lectura, escritura, creación, borrado y backup sobre todas
              las bases de datos.

Perfil:      RRHH
Usuarios:    GomezR, PerezB
Permisos:    Lectura y escritura sobre BD Empleados

Perfil:      RRHHBecario
Usuarios:    VargasM, PazO, GarciaG, CollJ
Permisos:    BD Empleados:  Lectura sobre nSalario de TEmpleado
                          Lectura y escritura sobre resto de tablas

Perfil:      Operaciones
Usuarios:    CasoA, FuertesG, GalaA, CastroR, LlamazaresJ, ...
Permisos:    BD Empleados:  Lectura sobre TEmpleado excepto nSalario
                          y cCuentaBancaria
              BD Operaciones: Lectura sobre TClientes
                          Lectura y escritura sobre resto de tablas

```

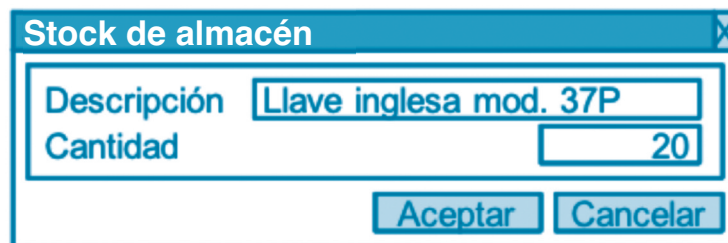
Figura 3.29. Definición de usuarios y perfiles en un SGBD

Se puede observar que, en el caso del personal técnico asociado con las bases de datos (administradores y operadores) se ha creado un usuario genérico asociado al perfil correspondiente, mientras los usuarios técnicos (personal de desarrollo) y no técnicos (personal de recursos humanos y operaciones) cuentan con un usuario por persona.

3.4.6. Accesos concurrentes

Un gran problema asociado a la edición de la información es el acceso simultáneo a los datos por parte de varios usuarios. Supongamos la siguiente situación:

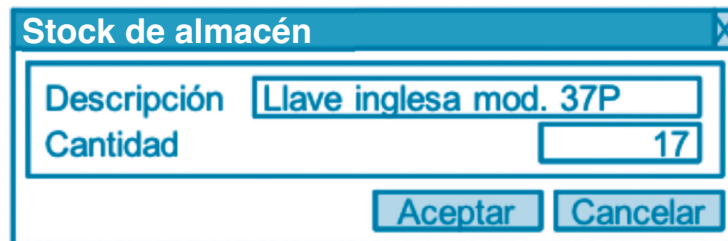
En un almacén de piezas para ferretería el empleado Primo recibe 10 unidades del producto “Llave inglesa mod. 37P”, por lo que entra a su aplicación de gestión de *stock* para que quede reflejado. Al abrir la pantalla de edición correspondiente al producto aparece lo siguiente:



Stock de almacén	
Descripción	Llave inglesa mod. 37P
Cantidad	20
Aceptar Cancelar	

Figura 3.30. Aplicación de *stock* de ferretería

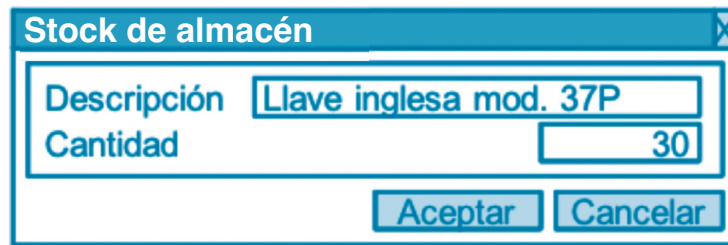
Mientras Primo apura su café, el empleado Segundo realiza una venta de 3 llaves inglesas modelo 37P, por lo que abre la aplicación y rápidamente las descuenta del stock:



Stock de almacén	
Descripción	Llave inglesa mod. 37P
Cantidad	17
Aceptar Cancelar	

Figura 3.31. Aplicación de *stock* de ferretería

Segundo presiona el botón “Aceptar”. En la base de datos se guardan correctamente las 17 llaves inglesas (las 20 que había menos las 3 que acaba de vender Segundo). Entonces Primo, una vez acabado su café, vuelve a su ordenador, donde siguen apareciendo 20 llaves inglesas en pantalla. Añade las 10 que acaba de recibir:



Stock de almacén

Descripción	Llave inglesa mod. 37P
Cantidad	30

Aceptar Cancelar

Figura 3.32
Aplicación de stock de ferretería

Primo presiona el botón “Aceptar”. En la base de datos, donde había 17 llaves inglesas ahora hay 30. Debería haber 27 (20 iniciales menos las 3 vendidas por Primo más las 10 recibidas por Segundo).

3.4.7. Políticas de bloqueo

Para solucionar este y otros problemas que se verán más adelante, el administrador de la base de datos debe implementar políticas de bloqueo, es decir, bloquear ciertos elementos de la base de datos mientras se esté efectuando una actualización que los afecte. Se pueden bloquear distintos elementos (la base de datos completa, una o varias tablas, bloques o páginas en memoria secundaria, columnas o registros). Bloquear parte de la base de datos impide su acceso por parte de otros usuarios, por lo que las políticas de bloqueo deben ser diseñadas con pragmatismo y precisión.

Fundamentalmente, se distingue entre dos tipos de bloqueos:

- *Compartidos*. Se puede consultar el valor del elemento bloqueado, pero no se puede modificar.
- *Exclusivos*. Se puede leer y modificar el valor del elemento bloqueado.

Es posible implementar varios bloqueos compartidos simultáneos sobre el mismo elemento, pero la casuística se complica cuando intervienen los bloqueos exclusivos, ya que solamente un proceso de actualización puede tener bloqueado un elemento de la base de datos. Por otro lado, varias peticiones consecutivas de bloqueo compartido sobre el mismo elemento podrían impedir que prosperase una solicitud de bloqueo exclusivo. En ese caso, se dice que dicha solicitud queda en estado de *inanición*. Los SGBD implementan diversas herramientas para luchar contra todos estos problemas.

Resumen

En los años setenta, E. F. Codd definió el modelo relacional, que flexibiliza los modelos de base de datos previos, y la teoría de la normalización, que elimina redundancias en el diseño de una base de datos.

Una vez que se cuenta con un diagrama entidad/relación, antes de llevarlo a la práctica es necesario convertirlo a modelo físico de datos, diseñando la estructura final de las tablas, campos y registros de la base de datos resultante. Para ello hay que aplicar una serie de reglas de transformación.

El modelo relacional también aborda los índices aplicados a las bases de datos, la definición de vistas, las restricciones sobre los campos de tablas, la definición de usuarios y perfiles de usuario, los criterios de integridad referencial, los accesos concurrentes y las políticas de bloqueo.

EJERCICIOS PROPUESTOS

1. Transformar la solución del ejercicio propuesto 1 del capítulo 2 (clientes de videoclub) en modelo físico de datos.
 2. Transformar la solución del ejercicio propuesto 2 del capítulo 2 (departamentos de empresa) en modelo físico de datos.
 3. Transformar la solución del ejercicio propuesto 3 del capítulo 2 (ampliación del ejercicio propuesto 2) en modelo físico de datos.
 4. Transformar la solución del ejercicio propuesto 4 del capítulo 2 (ferretería) en modelo físico de datos.
 5. Transformar la solución del ejercicio propuesto 5 del capítulo 2 (ampliación de la figura 2.21) en modelo físico de datos.
-

ACTIVIDADES DE AUTOEVALUACIÓN

1. Una relación con clave primaria simple:
 - ☐ a) Está en 1FN.
 - ☐ b) Está en 2FN.
 - ☐ c) Está en 2FN si está en 1FN.
2. Al pasar una relación M:N del DER al modelo físico de datos:
 - ☐ a) Hay que analizar las modalidades mínimas.
 - ☐ b) Dicha relación genera automáticamente una tabla.
 - ☐ c) Dicha relación nunca genera tabla.
3. Al pasar una relación 1:N (con modalidades 1..1 y 1..N) del DER al modelo físico de datos:
 - ☐ a) La clave primaria de la entidad con modalidad 1..N pasa a la entidad con modalidad 1..1.
 - ☐ b) La clave primaria de la entidad con modalidad 1..1 pasa a la entidad con modalidad 1..N.
 - ☐ c) Se genera una tabla de relación.
4. Todos los campos que conforman una clave primaria:
 - ☐ a) Deben contar con la restricción NOT NULL.
 - ☐ b) Deben contar con la restricción UNIQUE.
 - ☐ c) Deben contar con la restricción DEFAULT.
5. Cada perfil definido en una base de datos:
 - ☐ a) Corresponde a una persona física.
 - ☐ b) Corresponde a un solo usuario de la base de datos.
 - ☐ c) Puede corresponder a varios usuarios de la base de datos.

SOLUCIONES:

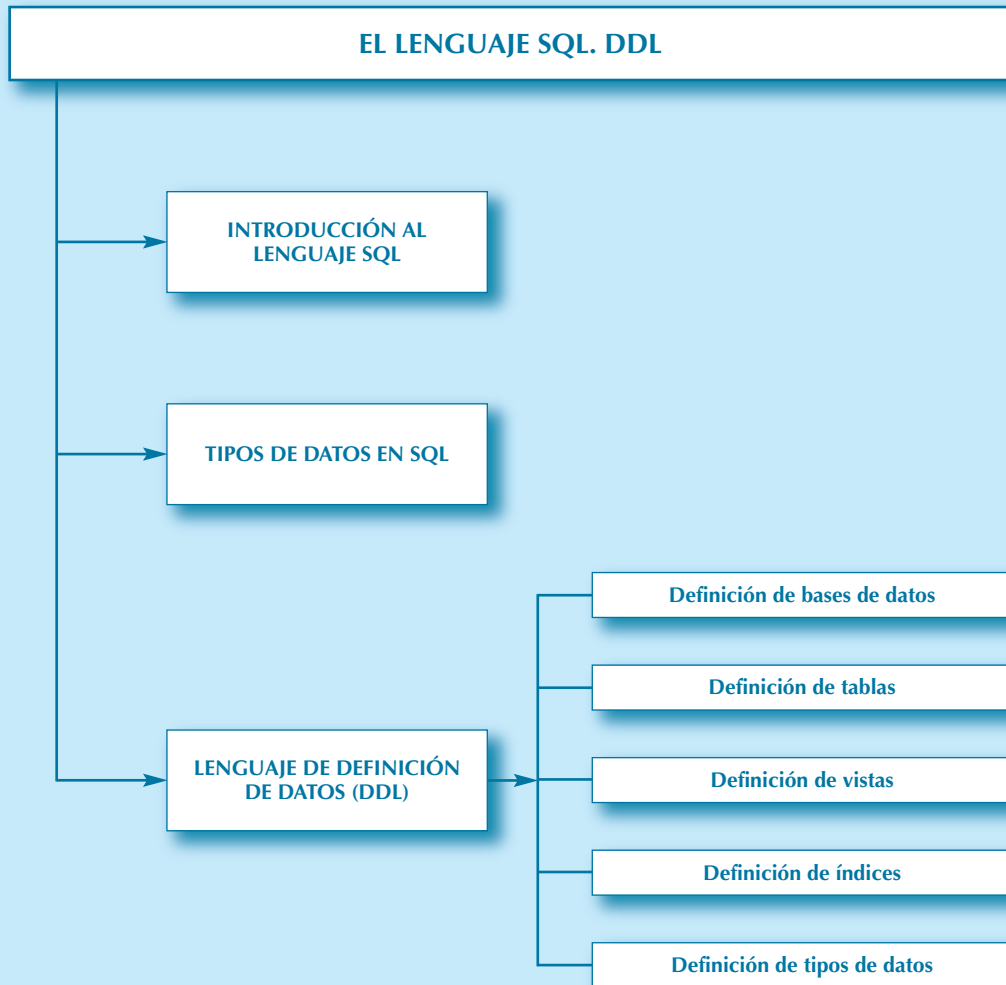
1. ☐ a ☐ b ☒ c2. ☐ a ☒ b ☐ c3. ☐ a ☒ b ☐ c4. ☒ a ☐ b ☐ c5. ☐ a ☐ b ☒ c

El lenguaje SQL. DDL

Objetivos

1. Conocer los condicionantes en que surgió el lenguaje SQL.
2. Comprender el ámbito y estructura del lenguaje SQL.
3. Familiarizarse con los tipos de datos del SQL estándar.
4. Crear, modificar y eliminar objetos de la base de datos mediante el DDL.

Mapa conceptual del capítulo



Glosario

Alfanumérico. Formato por letras, números y otros caracteres.

Álgebra relacional. Estudio de la organización de la información en conjuntos de datos, así como de los estados por los que transita dicha información.

Coma flotante. Notación científica para representar de forma reducida números reales muy grandes o muy pequeños.

ISO. Organización internacional de estándares (International Standards Organization). Organismo de estandarización.

4.1. Introducción al lenguaje SQL

Tras la exposición del modelo relacional por parte de E. F. Codd en 1970, dos compañeros del IBM Research Center, Donald D. Chamberlin y Raymond F. Boyce, definieron cuatro años más tarde un lenguaje de programación gracias al que interactuar con la información contenida en una base de datos. Dada la similitud de dicho lenguaje con la lengua cotidiana, lo llamaron SE-QUEL (*structured English query language*, lenguaje de consulta estructurado en inglés). Debido a problemas de *copyright*, el nombre se cambió a SQL.

Basado en la teoría del álgebra relacional y el cálculo relacional, SQL es un lenguaje de alto nivel con el que comunicarse con un SGBD(O)R. No solamente sirve para realizar consultas, altas, modificaciones y borrado de datos, sino también para definir la estructura de la base de datos, declarar sus objetos y elementos (usuarios, perfiles, *backups*, temporizaciones) y establecer elementos de seguridad.

Es importante ubicar SQL como lenguaje de programación:

- Es un lenguaje no procedimental, ya que se le especifica qué es lo que se quiere obtener sin entrar en el cómo (no se escriben instrucciones secuenciales y precisas, tan solo se indica qué acción se quiere realizar, y es tarea del propio lenguaje encontrar el camino adecuado para ejecutarla).
- Solo sirve para ejecutar consultas sobre bases de datos. Carece de estructuras de control (*if*, *while*, *for*), por lo que no es un lenguaje Turing completo.

La organización de estandarización ANSI formalizó SQL en 1986 (SQL-86). Desde entonces se han publicado otras siete versiones del estándar (SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008 y SQL:2011).

Las compañías que desarrollan SGBDR han ido ampliando notablemente el ámbito de SQL, acercándose y alejándose del estándar según la evolución de este y del mercado. En este sentido, han introducido una sintaxis propietaria que, si bien simplifica ciertas instrucciones u ofrece nuevas funcionalidades, dificulta la migración de bases de datos entre distintos sistemas gestores y crea dependencia por parte de sus clientes.

Las funcionalidades de SQL se suelen dividir en tres sublenguajes:

- DDL (*data definition language*, lenguaje de definición de datos). Permite crear la estructura de los elementos de la base de datos (tablas, índices, la propia base de datos).
- DML (*data manipulation language*, lenguaje de manipulación de datos). Opera sobre la información contenida en la base de datos (valores de los campos y registros). Al DML corresponde la inserción, modificación, borrado y consulta de los datos.
- DCL (*data control language*, lenguaje de control de datos). Se encarga de gestionar los permisos de los usuarios y los perfiles, así como la integridad de las transacciones sobre información de la base de datos.

Debido a la falta de universalidad de las implementaciones comerciales de SQL, así como a la rápida evolución del mercado, en este libro se seguirá la sintaxis SQL estándar. Conocido el estándar, la adaptación del lector a la sintaxis de cualquier otra versión de SQL será una tarea rápida y sencilla. Salvo que se indique lo contrario, se utilizará como referencia el estándar SQL:1999, ya que las versiones posteriores se centran en características que quedan fuera del ámbito de este texto. Se muestra una sintaxis simplificada en aras de garantizar un enfoque práctico.

PARA SABER MÁS



Alan Turing (1912-1954), uno de los padres de la informática, tuvo una importancia capital en el desarrollo de la Segunda Guerra Mundial, ya que descifró mensajes encriptados del ejército alemán. En 1936 definió su hipotética máquina universal de Turing, cuyo funcionamiento debe emular todo lenguaje de programación de cara a garantizar cierto poder computacional.

4.2. Tipos de datos en SQL

El estándar SQL define los siguientes tipos de datos, agrupados por familias:

- *Cadenas de caracteres*
 - CHAR (CHARACTER). Cadena de caracteres de longitud fija. Es necesario especificar dicha longitud. De lo contrario, se asumirá un único carácter.
Si la longitud del dato almacenado es inferior a la definida, se rellenará con espacios en blanco por la derecha, por lo que se suele utilizar como tipo de datos de campos

que almacenan información alfanumérica cuya longitud va a ser siempre idéntica – por ejemplo, se puede almacenar un NIF en un campo `cNIF` definido como `CHAR(9)`–.

- `VARCHAR` (`CHARACTER VARYING`). Cadena de caracteres con longitud variable. Es necesario especificar la longitud máxima.

Aunque la información presente en un campo de tipo `VARCHAR` sea inferior a su longitud máxima, solo se almacenarán los datos útiles. Se podría declarar un campo `cApellidos` como `VARCHAR(60)`, de modo que el valor “Sanz González” solo ocuparía 13 caracteres. Si dicho campo se hubiera declarado como `CHAR(60)`, se almacenarían los 60 caracteres, incluyendo 47 espacios en blanco a la derecha (“Sanz González”).

- `CLOB` (`CHARACTER LARGE OBJECT`). Cadena de caracteres de gran tamaño almacenada en un fichero distinto al de la tabla correspondiente. Algunos SGBDR limitan el uso de ciertas funcionalidades de trabajo con cadenas en este tipo de campos.

- *Números exactos*

Se deben utilizar para almacenar códigos (al ser su acceso más rápido que el de un campo alfanumérico) y valores numéricos operables. Un teléfono, por ejemplo, se debe almacenar en un campo de tipo cadena de caracteres, ya que, aunque está formado por números, no se puede utilizar para ejecutar una operación aritmética (no tiene sentido sumar o dividir teléfonos). Otro ejemplo aún más claro es el del código postal español. Está formado por cinco valores numéricos, pero no solo no es operable, sino que almacenar un código postal en un campo de tipo numérico desvirtuaría el dato, ya que el código postal “08080” de Barcelona quedaría registrado como 8080, perdiendo el cero a la izquierda que, en este caso, sí importa.

- `INT` (`INTEGER`). Número entero almacenado en 4 *bytes* (rango de valores entre –2.147.483.648 y 2.147.483.647).
- `SMALLINT`. Número entero almacenado en 2 *bytes* (rango de valores entre –32.768 y 32.767).
- `NUMERIC` o `DEC` (`DECIMAL`). Número con parte decimal de tamaño fijo. Se debe especificar el número de dígitos totales y el de dígitos decimales, de modo que un campo `nSalario` se definiría como `NUMERIC(8, 2)` –seis dígitos enteros y dos decimales–.

- *Números aproximados*

Ofrecen un rango mayor de representación de valores decimales, pero son menos precisos que el tipo `NUMERIC`, por lo que se recomienda utilizar este último para almacenar información muy sensible, como datos monetarios.

- `FLOAT` o `REAL`. Número real en coma flotante.
- `DOUBLE`. Número real en coma flotante con precisión doble.

- *Fechas, horas e intervalos*

- `DATE`. Fecha expresada en la secuencia año-mes-día (“yyyy-MM-dd”).
- `TIME`. Hora expresada en la secuencia hora-minuto-segundo (“hh:mm:ss”).

- *TIMESTAMP*. Almacena la fecha y hora de inserción o modificación del registro, en formato año-mes-día-hora-minuto-segundo (“yyyy-MM-dd hh:mm:ss”).
- *INTERVAL*. Representa un intervalo de tiempo, bien en la forma años-meses o en la forma días-horas-minutos-segundos (puede incluir fracciones de segundo).

PARA SABER MÁS

Generalmente, los SGBD almacenan las fechas como la suma de segundos transcurridos desde el 1 de enero de 1970. Es el formato conocido como *Unix time*.

- *Valores lógicos*
 - *BOOLEAN*. Incluye los valores “true” (verdadero) y “false” (falso), y puede almacenar el valor NULL en algunas implementaciones.
- *Objetos binarios*
 - *BLOB* (BINARY LARGE OBJECT). Se suele utilizar para almacenar imágenes, documentos u otros objetos, generalmente almacenados en ficheros distintos del de la tabla correspondiente.

El estándar SQL:2003 eliminó los tipos de datos BIT y BIT VARYING, e incluyó los tipos AUTO-GENERATEDVALUES y IDENTITY-COLUMNS, destinados a permitir que un campo que conforma la clave primaria de una tabla pueda generar sus valores de forma automática (generalmente incrementando el valor de ese campo en el último registro insertado), minimizando la intervención humana y añadiendo restricciones de seguridad. En varios SGBD, este tipo de campo se conoce como *autonumérico*.

- *Tipos definidos por el usuario*. Se verán en el siguiente apartado.

**Actividad propuesta 4.1**

Elegir los tipos de datos más adecuados para los campos de la tabla representada en la figura 3.1.

4.3. Lenguaje de definición de datos (DDL)

Las funcionalidades del lenguaje de definición de datos (*data definition language*, DDL) se limitan a la creación, modificación y borrado de los distintos objetos presentes en la base de datos, así como de la propia base de datos.

4.3.1. Definición de bases de datos

- Creación de una base de datos

```
CREATE DATABASE <base_datos>;
```

```
CREATE DATABASE BDBiblioteca;
```

Figura 4.1. Ejemplo de creación de la base de datos BDBiblioteca

- Borrado de una base de datos

```
DROP DATABASE <base_datos>;
```

```
DROP DATABASE BDBiblioteca;
```

Figura 4.2. Ejemplo de eliminación de la base de datos BDBiblioteca

4.3.2. Definición de tablas

- Creación de una tabla

```
CREATE TABLE <tabla> (
  <campo1> <tipo>[(<longitud>)] [NOT NULL][UNIQUE][PRIMARY KEY]
  [CHECK <condición>] [DEFAULT <valor>][, [
  <campo2> <tipo>[(<longitud>)] [NOT NULL][UNIQUE][PRIMARY KEY]
  CHECK <condición>][DEFAULT <valor>], ]
  ...
  <campoN> <tipo>[(<longitud>)] [NOT NULL][UNIQUE][PRIMARY KEY]
  [CHECK <condición>][DEFAULT <valor>]]
  [, PRIMARY KEY (<lista_campos>)]
  [, FOREIGN KEY (<lista_campos>) REFERENCES <tabla> (<lista_campos>)
  [{ON UPDATE [NO ACTION|SET DEFAULT|SET NULL|CASCADE]
  [ON DELETE [NO ACTION|SET DEFAULT|SET NULL|CASCADE]]
  }|
  {ON DELETE [NO ACTION|SET DEFAULT|SET NULL|CASCADE]
  [ON UPDATE [NO ACTION|SET DEFAULT|SET NULL|CASCADE]]
  }]]
  [, FOREIGN KEY...]
  ...
  [, FOREIGN KEY...]
);
```

```
CREATE TABLE TPais (
  nPaisID NUMERIC(3,0) NOT NULL UNIQUE PRIMARY KEY,
  cNombre VARCHAR(30) NOT NULL
);
```

Figura 4.3. Ejemplo de creación de la tabla TPais

Se asigna a nPaisID el tipo de datos NUMERIC al entender que se va a rellenar con la lista de códigos numéricos de país establecida por el organismo de estandarización ISO. Dado que dicha lista solo incluye códigos de tres números enteros, se especifican tres dígitos sin decimales.

```
CREATE TABLE TEditorial (  
    nEditorialID IDENTITY PRIMARY KEY,  
    cNombre VARCHAR(40) NOT NULL,  
    nPaisID NUMERIC(3,0) DEFAULT 724  
    , FOREIGN KEY (nPaisID) REFERENCES TPais (nPaisID)  
        ON UPDATE CASCADE  
        ON DELETE NO ACTION  
);
```

Figura 4.4

Ejemplo de creación de la tabla TEditorial con nPaisID como clave ajena de TPais

A nEditorialID se le ha asignado el tipo de datos IDENTITY. De ese modo, el SGBDR será responsable de asignarle valores válidos, únicos y correlativos. En cuanto a nPaisID, con función de clave ajena, en el caso de que se intente borrar un nPaisID de TPais, se impedirá dicho borrado; pero, en cambio, se permitirá la actualización, que se propagará a TEditorial en cascada. Nótese que nPaisID debe ser de tipo NUMERIC(3,0), porque así se definió en TPais, la tabla donde ejerce de clave primaria. Además, se le ha asignado un valor por defecto, de modo que, si se diese de alta una editorial sin país, el SGBD le asignaría automáticamente el código de país 724 (España).

```
CREATE TABLE TSocio (  
    cNIF CHAR(9) NOT NULL UNIQUE PRIMARY KEY,  
    cNombre VARCHAR(30) NOT NULL,  
    cApellidos VARCHAR(60) NOT NULL,  
    cDireccion VARCHAR(100),  
    cTelefono CHAR(12) NOT NULL,  
    dNacimiento DATE NOT NULL,  
    dAlta DATE NOT NULL CHECK dAlta >= "2003-09-01"  
);
```

Figura 4.5

Ejemplo de creación de la tabla TSocio

En este caso se ha decidido dar a cTelefono una longitud de 12 caracteres para poder incluir teléfonos internacionales. cDireccion es opcional, motivo por el que no lleva aparejada la cláusula NOT NULL. Mediante la cláusula CHECK se ha indicado una condición que debe cumplir el campo dAlta: que la fecha de alta de todo socio sea igual o superior al 1 de septiembre de 2003.

```
CREATE TABLE TPrestamo (  
    cSignatura VARCHAR(15) NOT NULL,  
    cNIF CHAR(9) NOT NULL,  
    dPrestamo DATE NOT NULL,  
    PRIMARY KEY (cSignatura, cNIF, dPrestamo)  
    , FOREIGN KEY (cSignatura) REFERENCES TEjemplar (cSignatura)  
    , FOREIGN KEY (cNIF) REFERENCES TSocio (cNIF)  
);
```

Figura 4.6. Ejemplo de creación de la tabla TPrestamo

Como la clave primaria de TPrestamo está formada por más de un campo, se define al final.

Actividad propuesta 4.2

Escribir las sentencias CREATE TABLE correspondientes a las tablas representadas en la figura 3.8 del capítulo anterior.



- *Modificación de una tabla.* La sentencia ALTER TABLE permite añadir un campo a una tabla, modificar sus características o borrarlo.

– Añadir un campo:

```
ALTER TABLE <tabla>  
    ADD <campo> <tipo>[( <longitud>)] [NOT NULL] [UNIQUE] [PRIMARY KEY]  
    [CHECK <condición>] [DEFAULT <valor>;]
```

```
ALTER TABLE TSocio  
    ADD cEmail VARCHAR(256);
```

Figura 4.7. Ejemplo de agregación del campo cEmail a TSocio

- Modificar un campo:

```
ALTER TABLE <tabla>  
    ALTER <campo> {TYPE <tipo>[( <longitud>)]|DROP DEFAULT};
```

La cláusula **TYPE** permite variar el tipo de datos o su longitud. **DROP DEFAULT** elimina el valor por defecto.

```
ALTER TABLE TSocio  
    ALTER cNombre VARCHAR(40);
```

Figura 4.8. Ejemplo de ampliación de longitud del campo TSocio.cNombre

- Borrar un campo:

```
ALTER TABLE <tabla>  
    DROP <campo>;
```

```
ALTER TABLE TLibro  
    DROP nAnyoPublicacion;
```

Figura 4.9. Ejemplo de eliminación de TLibro.nAnyoPublicacion

- Eliminación de una tabla.

```
DROP TABLE <tabla>;
```

```
DROP TABLE TTema;
```

Figura 4.10. Ejemplo de borrado de la tabla TTema



Actividad propuesta 4.3

Escribir las sentencias **ALTER TABLE** que permitan eliminar la fecha de nacimiento, incluir el correo electrónico y ampliar en cinco caracteres la dirección de los empleados de las tablas resultantes de la actividad propuesta 4.2.

4.3.3. Definición de vistas

Las vistas se basan en consultas efectuadas mediante la cláusula **SELECT**, que se verá con detalle en el próximo tema.

- *Creación de una vista*

```
CREATE VIEW <vista> (<lista_nombres_campos>)  
AS <cláusula_select>;
```

```
CREATE VIEW vSocios (Nombre, Apellidos, Direccion, Telefono)  
AS  
SELECT cNombre, cApellidos, cDireccion, cTelefono  
FROM TSocio;
```

Figura 4.11

Ejemplo de definición de una vista con información sobre socios. Nótese que los nombres de campo de la vista no tienen por qué coincidir con los de la tabla

- *Borrado de una vista*

```
DROP VIEW <vista>;
```

```
DROP VIEW vSocios;
```

Figura 4.12. Ejemplo de eliminación de la vista vSocios

4.3.4. Definición de índices

El estándar ANSI SQL no define la gestión de índices. A continuación se presenta una sintaxis genérica basada en la de varios SGBD(O)R comerciales (DB2, Oracle, PostgreSQL, MySQL y SQL Server).

- *Creación de un índice*

```
CREATE [UNIQUE] INDEX <índice>  
ON <tabla>  
(<campo1> [ASC|DESC][, <campo2> [ASC|DESC], ..., <campoN> [ASC|DESC]]);
```

Especificar **DESC** en un nombre de campo implica ordenar el índice de acuerdo a los valores de dicho campo de forma descendente. El valor por defecto es **ASC** (ordenación ascendente).

```
CREATE INDEX iSocio_Apellidos_Nombre
ON TSocio
(cApellidos, cNombre);
```

Figura 4.13. Ejemplo de definición de un índice sobre TSocio por apellidos y nombre

- Borrado de un índice

```
DROP INDEX <índice>;
```

```
DROP INDEX iSocio_Apellidos_Nombre;
```

Figura 4.14. Ejemplo de borrado del índice iSocio_Apellidos_Nombre



Actividad propuesta 4.4

Escribir la sentencia de creación de un índice sobre TSocio por fecha de alta (los socios más recientes primero), apellidos y nombre.

4.3.5. Definición de tipos de datos

El usuario de una base de datos puede crear tipos a medida que podrá reutilizar tantas veces como desee.

```
CREATE TYPE <nuevo_tipo> AS <tipo_estándar> [( <longitud> )];
```

```
CREATE TYPE typNombre AS
VARCHAR(30);
```

```
CREATE TYPE typCodPostal AS
CHAR(5);
```

Figura 4.15

Definición de tipos de datos creados por el usuario y aplicación de dichos tipos a la creación de una tabla. Nótese el uso del prefijo “typ” para continuar utilizando la notación húngara

```
CREATE TABLE TCliente (
    cNIF CHAR(9) NOT NULL UNIQUE PRIMARY KEY,
    cNombre typNombre NOT NULL,
    cDireccion VARCHAR(100),
    cCodPostal typCodPostal,
    cTelefono CHAR(12) NOT NULL
);
```

Actividad propuesta 4.5



Estudiar la base de datos de la figura 3.28 y decidir qué tipos de datos son candidatos a definirse explícitamente como tipos definidos por el usuario. Escribir las sentencias de creación de tipo correspondientes.

Resumen

El lenguaje SQL es la herramienta fundamental para interactuar con la información contenida en una base de datos relacional. Estandarizado por ANSI, permite al programador enviar instrucciones al SGBD sin necesidad de especificar paso a paso cómo debe llevar a cabo su tarea. ANSI también definió una serie de tipos de datos estándar que se asignan a los distintos campos de tablas relacionales.

SQL se divide en el lenguaje de definición de datos (DDL), el lenguaje de manipulación de datos (DML) y el lenguaje de control de datos (DCL), destinados a trabajar con objetos de la base de datos, con el contenido de dichos objetos y con la información de control, respectivamente.

El DDL permite crear, modificar y eliminar la base de datos y sus objetos (tablas, campos, vistas, índices).

EJERCICIOS PROPUESTOS

1. Buscar en Internet información sobre las novedades incorporadas por las distintas versiones del estándar SQL.
2. Elegir tipos de datos precisos para todos los campos de la base de datos de ejemplo del capítulo 3 (figura 3.28).
3. Escribir las sentencias **CREATE TABLE** correspondientes a todas las tablas de dicha base de datos.
4. Escribir las sentencias **ALTER TABLE** necesarias para incluir en la tabla Tautor los campos dNacimiento y dFallecimiento.
5. Escribir las sentencias de creación de índices obtenidas como resultado de la actividad propuesta 3.6.

ACTIVIDADES DE AUTOEVALUACIÓN

1. El lenguaje SQL:

- ☐ a) Carece de estructuras de control.
- ☐ b) Es un lenguaje Turing completo.
- ☐ c) Es un lenguaje de cuarta generación.

2. Un DNI se debe almacenar:

- ☐ a) En un campo de tipo numérico.
- ☐ b) En un campo de tipo alfanumérico.
- ☐ c) En un campo de tipo FLOAT o DOUBLE.

3. Para eliminar un elemento de la base de datos se utiliza la sentencia:

- ☐ a) REMOVE.
- ☐ b) DELETE.
- ☐ c) DROP.

4. La sentencia ALTER TABLE permite:

- ☐ a) Añadir o borrar campos de una tabla.
- ☐ b) Modificar campos de una tabla.
- ☐ c) Añadir, modificar y borrar campos de una tabla.

5. El tipo de datos IDENTITY:

- ☐ a) Permite definir una clave primaria.
- ☐ b) Permite que el campo que conforma la clave primaria genere sus valores automáticamente.
- ☐ c) Impide que un campo tenga valores repetidos.

SOLUCIONES:

1. ☒ a ☐ b ☐ c

2. ☐ a ☒ b ☐ c

3. ☐ a ☐ b ☒ c

4. ☐ a ☐ b ☒ c

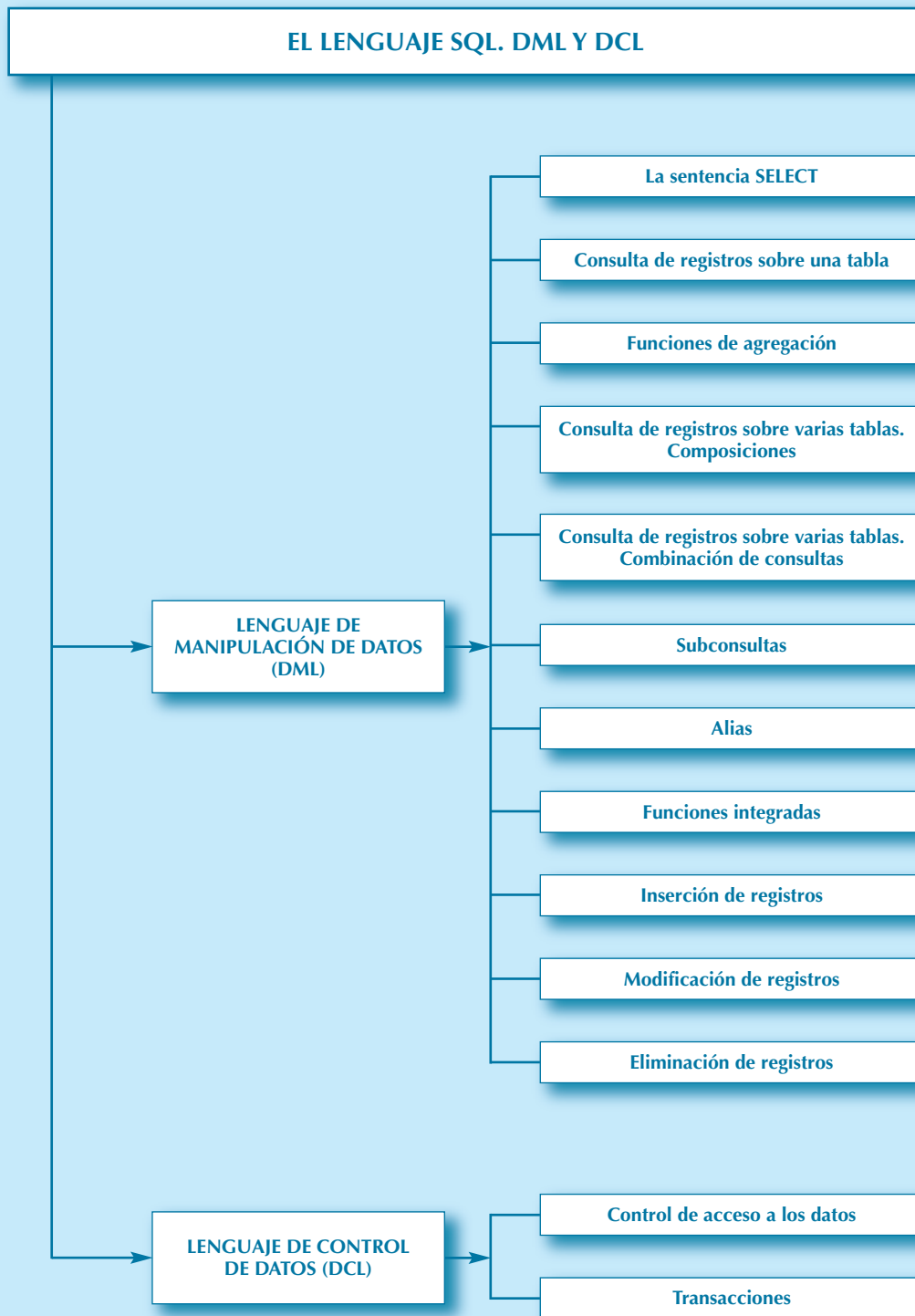
5. ☐ a ☒ b ☐ c

El lenguaje SQL. DML y DCL

Objetivos

1. Dominar la sintaxis de la sentencia **SELECT**.
2. Utilizar adecuadamente las funciones de agregación.
3. Aprender a realizar composiciones internas y externas.
4. Conocer las combinaciones de consultas y subconsultas.
5. Adquirir destreza en el uso de funciones integradas y alias.
6. Manejar la sintaxis del DML para inserciones, actualizaciones y borrados.
7. Entender cómo especifica el DCL la asignación de permisos de usuarios.
8. Familiarizarse con el concepto de transacción.

Mapa conceptual del capítulo



Glosario

Alias. Apodo o sobrenombre.

Calificar. Preceder el nombre de un elemento con el de otro elemento de mayor rango y un carácter separador.

Tabla de verdad. Cuadro utilizado para observar los resultados de expresiones lógicas.

Teoría de conjuntos. Campo de las matemáticas que estudia las relaciones entre colecciones de objetos.

Transacción. Conjunto de acciones sobre datos que deben ejecutarse de forma conjunta.

Valor límite. El que se encuentra en la frontera de un intervalo (p. ej., en el intervalo 3-28, tanto 3 como 28 son valores límite).

5.1. Lenguaje de manipulación de datos (DML)

El lenguaje de manipulación de datos (*data manipulation language*, DML) permite acceder a la información contenida en la base de datos. Mediante él se puede dar de alta nueva información y modificar o borrar la ya existente. No obstante, la faceta más utilizada del DML (y del lenguaje SQL) es la consulta de información, tarea de la que se encarga la sentencia **SELECT**.

5.1.1. La sentencia SELECT

```
SELECT [DISTINCT] <lista_campos>
FROM <lista_tablas>
[WHERE <condición>]
[GROUP BY <lista_campos>
  [HAVING <condición_de_agrupación>]]
[ORDER BY <lista_campos>];
```

<lista_tablas> puede incluir una o varias tablas. Primero veremos cómo implementar una sentencia **SELECT** sobre una sola tabla.

5.1.2. Consulta de registros sobre una tabla

`<lista_campos>` especifica la lista de campos o expresiones a obtener separados por comas y en el orden especificado. Se puede utilizar el comodín *, que identifica todos los campos de la tabla. `DISTINCT` elimina los valores repetidos.

`<condición>` puede ser una condición simple o la concatenación de varias de ellas separadas mediante operadores lógicos. Las condiciones generalmente comparan el valor de un campo con el de una variable, otro campo o una expresión. Vemos la lista de operadores disponibles:

- *De comparación:* `<`, `<=`, `>`, `>=`, `<>`, `=`. Comparan dos valores, típicamente el valor de un campo con un valor fijo, expresión o valor de otro campo.

```
SELECT cNombre, cApellidos
FROM TSocio
WHERE dAlta = "2013-10-03";
```

Figura 5.1

Consulta de nombre y apellido de todos los socios dados de alta el 3 de octubre de 2013

```
SELECT *
FROM TLibro
WHERE nAnyoPublicacion >= 2010;
```

Figura 5.2

Consulta de los valores de todos los campos de los libros publicados desde 2010 inclusive

El operador “distinto de” (`<>`) en ciertas implementaciones se especifica como “no igual” (`!=`).

- `BETWEEN ... AND`. Establece una comparación dentro de un intervalo cerrado (incluyendo los valores límite).

```
SELECT cNombre, cApellidos
FROM TSocio
WHERE dAlta BETWEEN "2011-01-01" AND "2011-12-31";
```

Figura 5.3

Consulta de nombre y apellidos de los socios dados de alta en 2011

- **Lógicos: AND, OR, NOT.** La teoría del álgebra *booleana* aplicada a un contexto de programación informática establece los siguientes usos de los operadores lógicos:
 - Dos expresiones lógicas unidas por el operador **AND** se evaluarán a cierto (*true*) si ambas son ciertas.

CUADRO 5.1. Tabla de verdad para el operador AND

AND	Cierto	Falso
Cierto	Cierto	Falso
Falso	Falso	Falso

- Para que dos expresiones lógicas unidas por el operador **OR** se evalúen a cierto basta con que una de ellas sea cierta.

CUADRO 5.2. Tabla de verdad para el operador OR

OR	Cierto	Falso
Cierto	Cierto	Cierto
Falso	Cierto	Falso

- Una expresión lógica precedida por el operador **NOT** se evalúa al valor opuesto (si la expresión es cierta se convertirá en falsa *¬false* y viceversa).

```
SELECT *
  FROM TSocio
 WHERE cNombre = "Javier"
        AND cApellidos = "Peinado Martín";
```

Figura 5.4. Consulta de los valores de todos los campos para el socio o socios de nombre "Javier" y apellidos "Peinado Martín". Ambas condiciones deben cumplirse

```
SELECT DISTINCT nAnyoPublicacion
  FROM TLibro
 WHERE cTitulo = "The Tempest"
        AND nEditorialID <> 16;
```

Figura 5.5. Consulta del año de publicación de todos los libros de título "The Tempest", excepto los de la editorial 16 y excluyendo los años duplicados

```
SELECT cNIF, cNombre, cApellidos
FROM TSocio
WHERE dAlta < "2000-01-01"
      OR dAlta >= "2010-01-01";
```

Figura 5.6. Consulta del NIF, el nombre y los apellidos de los socios dados de alta antes de 2000 y de los dados de alta de 2010 en adelante

```
SELECT cSignatura
FROM TPrestamo
WHERE NOT (cNIF = "09653801B"
          AND dPrestamo = "2007-05-13");
```

Figura 5.7. Consulta de la signatura de los ejemplares de libros prestados que no se hayan entregado al socio de NIF "09653801B" el 13 de mayo de 2007

Cuando se combinan operadores AND y OR el primero tiene precedencia sobre el segundo, es decir, se evalúa antes. En la expresión $A \text{ AND } B \text{ OR } C$ primero se evaluaría $A \text{ AND } B$ y su resultado se evaluaría en un OR con C . En cualquier caso cuando ambos operadores conviven es recomendable utilizar siempre paréntesis para establecer el orden de precedencia de forma explícita. La expresión comentada quedaría como $(A \text{ AND } B) \text{ OR } C$, cuya evaluación arroja un resultado muy distinto de la expresión $A \text{ AND } (B \text{ OR } C)$.

PARA SABER MÁS



El álgebra booleana se debe a los trabajos del matemático inglés George Boole en el campo de la lógica deductiva. Posteriormente, el matemático e ingeniero estadounidense Claude Shannon la aplicó al diseño de circuitos.

- *De concatenación.* El operador `||` concatena cadenas de caracteres (provenientes de campos o especificadas por el usuario), y permite mostrar información de varias fuentes en una sola expresión.

```
SELECT cNIF || " - " || cApellidos || ", " || cNombre
FROM TSocio;
```

```
00789521T - Sanz González, Paula
50687452Y - García Viñals, José Luis
38546998X - Peinado Martín, Javier
...
```

Figura 5.8

Consulta de nombre y apellidos de los socios y resultado de la consulta

En muchos SGBD comerciales el operador de concatenación es el símbolo “+”.

- **IS NULL.** Comprueba si un valor determinado es nulo. Se puede combinar con **NOT** en la forma **IS NOT NULL**.

```
SELECT *
FROM TSocio
WHERE cDireccion IS NOT NULL;
```

Figura 5.9. Consulta de los datos de los socios que tienen dirección

Es necesario recordar en este punto la diferencia entre un valor nulo y un valor vacío (0 en campos numéricos y “” –cadena vacía– en campos de tipo cadena de caracteres). La siguiente consulta provocaría un error si el valor de **cDireccion** fuera nulo, algo que no ocurriría si dicho valor fuese una cadena vacía:

```
SELECT cApellidos || ", " || cNombre || ". Dirección: " || cDireccion
FROM TSocio
WHERE cNIF = "50687452Y";
```

Figura 5.10. Ejemplo de concatenación con valores nulos

- **LIKE.** Compara cadenas de caracteres de forma aproximada (es decir, sin que tenga que coincidir su contenido exacto), mediante el uso de los comodines **%** (sustituye a cualquier cadena de caracteres) y **_** (sustituye a un único carácter). Para usos simples de este operador se pueden definir las siguientes reglas:

CUADRO 5.3. Uso básico del operador LIKE

Texto que empiece por una cadena	<code><campo> LIKE "<cadena>%"</code>
Texto que acabe por una cadena	<code><campo> LIKE "%<cadena>"</code>
Texto que contenga una cadena	<code><campo> LIKE "%<cadena>%"</code>

```
SELECT *
  FROM TSocio
 WHERE cApellidos LIKE "Garc_a R%guez";
```

García Rodríguez	García Rodriguez	García Rguez
Garcia Rodríguez	Garcia Rodriguez	Garcia Rguez

Figura 5.11
Consulta que busca distintas representaciones de la secuencia de apellidos “García Rodríguez” y posibles resultados encontrados

En algunos SGBD(O)R comerciales, en vez de % y _ se utilizan los comodines * y ?, respectivamente.



Actividad propuesta 5.1

Escribir las sentencias SQL que ejecuten las siguientes consultas sobre la base de datos de la figura 3.28:

- a) Listar los socios de nombre “Jacinto” nacidos antes de 1970 que se dieron de alta como socios de la biblioteca en la década de los noventa.
- b) Obtener los libros que no hayan sido publicados por las editoriales de código 15 o 32, o bien los publicados antes del año 2000.
- c) Mostrar el nombre y los apellidos de los socios que tienen teléfono, pero no dirección.
- d) Listar los autores de apellido “Byatt” cuyo nombre empiece por una “A” (mayúscula) y contenga una “S” (mayúscula).

5.1.3. Funciones de agregación

<lista_campos> también puede contener expresiones obtenidas como resultado de la ejecución de funciones de agregación, es decir, cálculos resultantes de agregar los registros afectados por la consulta. Son las siguientes:

- `COUNT(*|nombre_campo)`. Número de registros que cumplen la condición. Si en vez del comodín `*` se especifica un nombre de campo, se excluyen los registros cuyo valor sea `NULL` para ese campo.

```
SELECT COUNT(*)  
  FROM TSocio  
 WHERE dAlta BETWEEN "2012-01-01" AND "2012-12-31";
```

Figura 5.12. Consulta del número de socios dados de alta en 2012

```
SELECT COUNT(cDireccion)  
  FROM TSocio;
```

Figura 5.13

Consulta del número de socios que tienen dirección. Obtiene el mismo resultado que la consulta `SELECT COUNT(*) FROM TSocio WHERE cDireccion IS NOT NULL;`

- `AVG(campo)`. Media aritmética de un campo numérico o intervalo.

```
SELECT AVG(nSalario)  
  FROM TEmpleado;
```

Figura 5.14. Consulta de salario medio de una tabla de empleados

- `SUM(campo)`. Suma de valores de un campo numérico o intervalo.

```
SELECT SUM(nSalario)  
  FROM TEmpleado  
 WHERE nDepartamentoID = 3;
```

Figura 5.15. Cálculo de la suma de salarios de un departamento

- `MAX(campo)`. Valor máximo de un campo.
- `MIN(campo)`. Valor mínimo de un campo.

```
SELECT MAX(dAlta), MIN(dNacimiento)
FROM TSocio;
```

Figura 5.16

Consulta de la fecha de alta del socio más reciente
y la de nacimiento del más anciano

Si en una consulta queremos recuperar los valores de otros campos junto con el resultado de una función agregada, hemos de hacer que el resultado de dicha función se refiera a cada combinación exclusiva de valores de los campos que se van a obtener. Eso se consigue mediante la cláusula `GROUP BY`:

```
SELECT <lista_campos>, <funcion_agregación>
FROM <tabla>
GROUP BY <lista_campos>;
```

Es obligatorio que todos los campos que aparecen en el `SELECT` también aparezcan en el `GROUP BY`.

```
SELECT nAnyoPublicacion, COUNT(*)
FROM TLibro
GROUP BY nAnyoPublicacion;
```

nAnyoPublicacion	COUNT(*)
2014	12
2008	35
1997	4
2009	75
...	

Figura 5.17

Consulta del número de libros publicados por año, y su resultado

```
SELECT TSocio.cNombre, TSocio.cApellidos, COUNT(*)
FROM TSocio INNER JOIN TPrestamo
ON TSocio.cNIF = TPrestamo.cNIF
GROUP BY TSocio.cNombre, TSocio.cApellidos;
```

cNombre	cApellidos	COUNT(*)
Paula	Sanz González	3
José Luis	García Viñals	24
Javier	Peinado Martín	11
...		

Figura 5.18

Consulta del número de libros tomados en préstamo por cada socio, y su resultado

```
SELECT cTitulo, nEditorialID, MIN(nAnyoPublicacion)
FROM TLibro
GROUP BY cTitulo, nEditorialID;
```

cTitulo	nEditorialID	MIN(nAnyoPublicacion)
Laura y Julio	32	2006
Rayuela	35	1963
The Tempest	102	1942
...		

Figura 5.19

Consulta, por cada libro y editorial, del año de publicación más antiguo, y su resultado

La cláusula de agrupación **HAVING** especifica una condición sobre el resultado de una agrupación, y actúa sobre **GROUP BY** de forma análoga al funcionamiento de la cláusula **WHERE** sobre una consulta no agregada.

```
SELECT cNIF, COUNT(*)
FROM TPrestamo
GROUP BY cNIF
HAVING COUNT(*) > 3;
```

cNIF	COUNT(*)
50687452Y	24
38546998X	11
...	

Figura 5.20. Consulta de los NIF de todos los socios que se han llevado un mínimo de cuatro libros prestados, junto con el número de libros que cada uno ha sacado de la biblioteca, y su resultado

```
SELECT cTitulo, nEditorialID, MIN(nAnyoPublicacion)
FROM TLibro
GROUP BY cTitulo, nEditorialID
HAVING MIN(nAnyoPublicacion) < 1950;
```

cTitulo	nEditorialID	MIN(nAnyoPublicacion)
The Tempest	102	1942
...		

Figura 5.21. Consulta, por cada libro y editorial, del año de publicación más antiguo, solamente para libros publicados antes de 1950, y su resultado

GROUP BY y HAVING se pueden combinar con la cláusula WHERE, que filtrará la información de forma previa a su agregación.

```
SELECT cTitulo, nEditorialID, MIN(nAnyoPublicacion)
FROM TLibro
WHERE nEditorialID <> 32
GROUP BY cTitulo, nEditorialID
HAVING MIN(nAnyoPublicacion) < 1950;
```

cTitulo	nEditorialID	MIN(nAnyoPublicacion)
The Tempest	102	1942
...		

Figura 5.22. Consulta de la figura 5.21 excluyendo los libros de la editorial 32, y su resultado

Por otro lado, la cláusula **ORDER BY** especifica por qué campos se van a ordenar los registros resultado de la consulta, y si dicho orden va a ser ascendente o descendente para cada uno de esos campos (por defecto es ascendente). De cara a optimizar el rendimiento de la base de datos, es importante que el orden especificado coincida con el de los campos de un índice existente.

```
SELECT nEditorialID, cTitulo, nAnyoPublicacion
FROM TLibro
ORDER BY nEditorialID, nAnyoPublicacion DESC;
```

Figura 5.23

Consulta de libros ordenados según la editorial y, dentro de cada editorial, descendientemente por año de publicación (el más reciente primero)

Actividad propuesta 5.2



Escribir las sentencias SQL que ejecuten las siguientes consultas sobre la base de datos de la figura 3.28:

- Calcular el número de libros publicados en 1993 por la editorial 32.
- Mostrar el número de libros tomados en préstamo por el socio de NIF "50687452Y" por cada día del año 2014.
- Modificar la consulta anterior para que solo aparezcan los días en que el socio especificado se llevó un mínimo de tres libros.
- Listar todos los socios de la biblioteca desde el más reciente al más antiguo. Los socios que se dieran de alta el mismo día aparecerán ordenados alfabéticamente (por apellidos y nombre) dentro de ese día.

5.1.4. Consulta de registros sobre varias tablas. Composiciones

El lenguaje SQL permite enlazar varias tablas en una consulta mediante una operación llamada composición o **JOIN**. La sintaxis definida en el estándar SQL-86 (y vigente en algunos SGBDR comerciales) es la siguiente:

```
SELECT [DISTINCT] <lista_campos>
FROM <tabla1>, <tabla2>[,...,<tablaN>]
WHERE <condición_de_relación>;
```

En los siguientes ejemplos se muestra una composición de dos y tres tablas, respectivamente:

```
SELECT cSignatura, cTitulo
FROM TEjemplar, TLibro
WHERE TEjemplar.nLibroID = TLibro.nLibroID;
```

cSignatura	cTitulo
N MIL lau	Laura y Julio
N COR ray	Rayuela
T SHA tem	The Tempest
...	

Figura 5.24

Consulta de la lista de firmas junto con el nombre del libro correspondiente, y su resultado

```
SELECT cSignatura, cTitulo, cNombre
FROM TEjemplar, TLibro, TEditorial
WHERE TEjemplar.nLibroID = TLibro.nLibroID
AND TLibro.nEditorialID = TEditorial.nEditorialID;
```

cSignatura	cTitulo	cNombre
N MIL lau	Laura y Julio	Alfaguara
N COR ray	Rayuela	Bruguera
T SHA tem	The Tempest	Penguin
...		

Figura 5.25

Ampliación de la consulta anterior incluyendo el nombre de la editorial, y su resultado

Al introducir JOIN es posible que encontremos ambigüedad en los nombres de campo, ya que dos campos con el mismo nombre pueden darse en tablas distintas que forman parte de un JOIN. En ese caso, es necesario calificar el nombre del campo con el de la tabla, y se considera una recomendación de estilo calificar el nombre de todos los campos que intervengan en un JOIN, se presten o no a duplicidades.

```
SELECT TAuthor.cNombre, TPais.cNombre
FROM TAuthor, TAuthorPais, TPais
WHERE TAuthor.nAutorID = TAuthorPais.nAutorID
      AND TAuthorPais.nPaisID = TPais.nPaisID
ORDER BY TAuthor.cNombre;
```

Figura 5.26. Consulta de autores junto con sus nacionalidades

Actividad propuesta 5.3



Escribir las sentencias SQL que ejecuten las siguientes consultas sobre la base de datos de la figura 3.28:

- Listar el título de los libros de la editorial 32 junto con su tema o temas.
- Mostrar el nombre y apellidos de todos los autores con libros publicados junto con el número de libros que tienen en la biblioteca.
- Mostrar el nombre y apellidos de todos los autores con libros publicados junto con el año de publicación más antiguo de todos sus libros.
- Por cada signatura y fecha de préstamo, mostrar el título del libro correspondiente y el nombre y apellidos del socio que se lo llevó en préstamo.

El estándar SQL-92 incluyó nuevas funcionalidades relacionadas con JOIN, así como una nueva nomenclatura que parte de la expresión INNER JOIN.

```
SELECT TEjemplar.cSignatura, TLibro.cTitulo, TEditorial.cNombre
FROM TEjemplar
      INNER JOIN TLibro
            ON TEjemplar.nLibroID = TLibro.nLibroID
      INNER JOIN TEditorial
            ON TLibro.nEditorialID = TEditorial.nEditorialID;
```

Figura 5.27. Consulta de signaturas, libros y editoriales con la sintaxis INNER JOIN

Actividad propuesta 5.4



Repetir las consultas de la actividad propuesta 5.3 con la sintaxis INNER JOIN.

Un problema presentado por el antiguo **JOIN** es su exclusión de registros no relacionados, motivo por el que su funcionalidad es conocida como *composición interna*. Supongamos los siguientes datos:

TSocio		
cNIF	cNombre	cApellidos
00789521T	Paula	Sanz González
50687452Y	José Luis	García Viñals
38546998X	Javier	Peinado Martín

TEjemplar		
cSignatura	nLibroID	
68 LOP bas	37	
68 PRE ing	41	
687 CAB ana	15	

TPrestamo			
cSignatura	cNIF	dPrestamo	
68 PRE ing	38546998X	2012-06-23	
68 PRE ing	00789521T	2013-12-21	
687 CAB ana	00789521T	2013-12-21	

Figura 5.28. Información sobre préstamos de libros

Si quisiéramos ver la lista de préstamos de ejemplares a socios, ejecutaríamos la siguiente consulta:

```
SELECT TPrestamo.cSignatura, TSocio.cNombre, TSocio.cApellidos
FROM TEjemplar, TPrestamo, TSocio
WHERE TEjemplar.cSignatura = TPrestamo.cSignatura
AND TPrestamo.cNIF = TSocio.cNIF;
```

cSignatura	cNombre	cApellidos
68 PRE ing	Javier	Peinado Martín
68 PRE ing	Paula	Sanz González
687 CAB ana	Paula	Sanz González

Figura 5.29

Composición interna de préstamos de ejemplares a socios y resultado de la consulta

Nótese que no aparecen datos de “José Luis García Viñals”, al no haber tomado ningún libro en préstamo, ni del ejemplar con signatura “68 LOP bas”, que nadie se llevó.

Entre las nuevas funcionalidades del estándar SQL-92 se encuentra la *composición externa* o OUTER JOIN, que permite incluir entre los resultados de la consulta los registros de alguna o de ambas tablas que no tengan relación. Hay tres subtipos:

- **LEFT (OUTER) JOIN.** Incluye en los resultados los registros de la tabla situada a la izquierda de la consulta sin relación con registros a la derecha.

```
SELECT TEjemplar.cSignatura, TSocio.cNombre, TSocio.cApellidos
FROM TEjemplar
LEFT JOIN TPrestamo
    ON TEjemplar.cSignatura = TPrestamo.cSignatura
LEFT JOIN TSocio
    ON TPrestamo.cNIF = TSocio.cNIF;
```

cSignatura	cNombre	cApellidos
68 LOP bas	NULL	NULL
68 PRE ing	Javier	Peinado Martín
68 PRE ing	Paula	Sanz González
687 CAB ana	Paula	Sanz González

Figura 5.30

Composición externa de préstamos de ejemplares y resultado de la consulta. Aparece el ejemplar de signatura “68 LOP bas”, aunque ningún socio lo haya tomado en préstamo

- **RIGHT (OUTER) JOIN.** Incluye en los resultados los registros de la tabla situada a la derecha de la consulta sin relación con registros a la izquierda.

```
SELECT TEjemplar.cSignatura, TSocio.cNombre, TSocio.cApellidos
FROM TEjemplar
RIGHT JOIN TPrestamo
    ON TEjemplar.cSignatura = TPrestamo.cSignatura
RIGHT JOIN TSocio
    ON TPrestamo.cNIF = TSocio.cNIF;
```

cSignatura	cNombre	cApellidos
68 PRE ing	Javier	Peinado Martín
68 PRE ing	Paula	Sanz González
687 CAB ana	Paula	Sanz González
NULL	José Luis	García Viñals

Figura 5.31. Composición externa de préstamos a socios y resultado de la consulta. Aparece el socio José Luis García Viñals, aunque no haya tomado ningún libro en préstamo

- **FULL (OUTER) JOIN.** Incluye en los resultados todos los registros, independientemente de si cuentan con registros relacionados en otras tablas.

```
SELECT TPrestamo.cSignatura, TSocio.cNombre, TSocio.cApellidos
FROM TEjemplar
    OUTER JOIN TPrestamo
        ON TEjemplar.cSignatura = TPrestamo.cSignatura
    OUTER JOIN TSocio
        ON TPrestamo.cNIF = TSocio.cNIF;
```

cSignatura	cNombre	cApellidos
68 LOP bas	NULL	NULL
68 PRE ing	Javier	Peinado Martín
68 PRE ing	Paula	Sanz González
687 CAB ana	Paula	Sanz González
NULL	José Luis	García Viñals

Figura 5.32. Composición externa de préstamos de ejemplares a socios y resultado de la consulta. Aparecen tanto los ejemplares no tomados en préstamo como los socios que no se han llevado ningún ejemplar aún



Actividad propuesta 5.5

Escribir las sentencias SQL que ejecuten las siguientes consultas sobre la base de datos de la figura 3.28:

- Listar todos los nombres de tema junto con los títulos de los libros asociados a cada tema. Da igual que se repitan temas, pero deben aparecer todos (incluso los temas para los que no hay libros en la biblioteca). Ordenar por nombre de tema.
- Mostrar los nombres y apellidos de todos los socios dados de alta en el año 2013 junto con el título de los libros que tomaron en préstamo durante dicho año. Deben aparecer todos los socios dados de alta en 2013, incluso aunque no se hayan llevado libros en préstamo. Ordenar por apellidos y nombre de socio.
- Consultar el nombre y apellidos de todos los autores junto con su nacionalidad o nacionalidades y los títulos de sus libros en esta biblioteca. Deben aparecer todos los autores, aunque no tengan libros en la biblioteca. Ordenar por nombre y apellidos de autor.

5.1.5. Consulta de registros sobre varias tablas. Combinación de consultas

Se puede combinar el resultado de varias consultas SQL independientes mediante los siguientes operadores:

- **UNION [ALL]**. Unión de conjuntos (todos los datos que retorna una consulta más todos los datos que retorna otra). Por defecto se excluyen los resultados repetidos, a no ser que se especifique la cláusula **ALL**.

```
SELECT cTitulo
      FROM TLibro
      UNION
SELECT cTitulo
      FROM BDCatalogoBibliotecaNacional.TLibro;
```

Figura 5.33

Consulta de todos los libros y, además, los existentes en la base de datos BDCatalogoBibliotecaNacional

Nótese que para referenciar una tabla de otra base de datos basta con prefijar el nombre de la tabla con el de la base de datos y un punto.

- **MINUS**. Diferencia de conjuntos (todos los datos que retorna una consulta excepto los que retorna la otra).

```
SELECT cTitulo
      FROM BDCatalogoBibliotecaNacional.TLibro
      MINUS
SELECT cTitulo
      FROM TLibro;
```

Figura 5.34

Consulta de los libros del catálogo de la Biblioteca Nacional que no existen en nuestra biblioteca

- **INTERSECT**. Intersección de conjuntos (todos los datos que figuran en el resultado de ambas consultas).

```
SELECT cTitulo
      FROM TLibro
      INTERSECT
SELECT cTitulo
      FROM BDCatalogoBibliotecaNacional.TLibro;
```

Figura 5.35

Consulta de los títulos que están tanto en nuestra biblioteca como en el catálogo de la Biblioteca Nacional



Actividad propuesta 5.6

Escribir las sentencias SQL que ejecuten las siguientes consultas sobre la base de datos de la figura 3.28:

- Mostrar el título de los libros publicados a la vez en 1970 y 1989, entendiendo que estamos hablando de distintas ediciones de la misma obra.
- Listar los apellidos y nombre de los socios dados de alta en 2012 seguidos de los apellidos y nombre de los socios dados de alta en 2013.

5.1.6. Subconsultas

SQL también permite combinar consultas mediante la evaluación de valores con los resultados de subconsultas.

Se puede anidar un número ilimitado de subconsultas. La relación se establece, entre otros, mediante los siguientes operadores:

- Operadores de comparación. La subconsulta debe devolver un solo valor.

```
SELECT cSignatura
      FROM TPrestamo
      WHERE cNIF = (SELECT cNIF
                    FROM TSocio
                    WHERE cApellidos = "Sanz González"
                    AND cNombre = "Paula");
```

Figura 5.36

Consulta de la signatura de los ejemplares prestados a "Paula Sanz González"

```
SELECT cTitulo
FROM TLibro
WHERE nAnyoPublicacion < (SELECT MIN(nAnyoPublicacion)
                           FROM TLibro
                           WHERE cNombre = "Alfaguara");
```

Figura 5.37

Consulta del título de los libros cuyo año de publicación sea inferior al libro más antiguo de la editorial “Alfaguara”

- IN. Comprueba la pertenencia de un valor a los valores devueltos por una subconsulta.

```
SELECT cSignatura
FROM TEjemplar
WHERE nLibroID IN (SELECT nLibroID
                   FROM TLibro
                   WHERE nEditorialID = 32);
```

Figura 5.38

Consulta de signatura de los ejemplares correspondientes a libros de la editorial 32

Dado que el operador IN trata el resultado de la subconsulta como una lista de valores, la comparación no se establece contra un valor concreto, por lo que no hay problemas con los valores nulos.

PARA SABER MÁS



Un error habitual consiste en entender las subconsultas como un sustituto de las consultas a varias tablas con una sintaxis simplificada. Este enfoque no es recomendable debido a que el uso de subconsultas suele presentar serios problemas de rendimiento, ya que la consulta principal comparará cada uno de los posibles registros con el resultado de la subconsulta (en el ejemplo de la figura 5.38, recorrerá TEjemplar y para cada registro comparará el valor de nLibroID con el devuelto por la subconsulta). Además, si el valor devuelto por una subconsulta es NULL y el operador de comparación elegido espera un único resultado, la consulta generará un error.



Actividad propuesta 5.7

Escribir las sentencias SQL que ejecuten las siguientes consultas con subconsultas sobre la base de datos de la figura 3.28:

- Consultar el nombre y apellidos del socio más antiguo.
- Mostrar el número de libros por año de publicación pertenecientes a editoriales de países que cuenten con, al menos, tres autores.

5.1.7. Alias

Los alias permiten asignar nombres alternativos a campos, expresiones de consulta o tablas completas. Recordemos la consulta de la figura 5.8:

```
SELECT cNIF || " - " || cApellidos || ", " || cNombre
FROM TSocio;
```

```
cNIF || " - " || cApellidos || ", " || cNombre
-----
00789521T - Sanz González, Paula
50687452Y - García Viñals, José Luis
38546998X - Peinado Martín, Javier
```

Figura 5.39. Consulta de socios y salida por pantalla

Como cabecera de los datos de salida aparece la expresión de concatenación. Mediante la palabra **AS** se puede asociar a dicha expresión un alias más legible para simplificar su salida por pantalla:

```
SELECT cNIF || " - " || cApellidos || ", " || cNombre AS "Socio"
FROM TSocio;
```

```
Socio
-----
00789521T - Sanz González, Paula
50687452Y - García Viñals, José Luis
38546998X - Peinado Martín, Javier
```

Figura 5.40. Consulta de socios con alias y salida por pantalla

Los alias también se pueden asociar a nombres de tablas, de modo que en una consulta podamos utilizar la misma tabla más de una vez con distintos fines. En el siguiente ejemplo se muestran los libros de la biblioteca junto con sus autores y editorial. También aparecerá la nacionalidad de los autores y el país donde reside cada editorial, por lo que serán necesarios dos JOIN independientes a TPais. Dicha independencia se basa en la asociación de una de las dos apariciones de TPais a un alias (TPaisEditorial):

```
SELECT TLibro.cTitulo AS "Título",
       TAutor.cNombre || " " || TAutor.cApellidos ||
       " (" || TPais.cNombre || ")" AS "Autor",
       TEditorial.cNombre ||
       " (" || TPaisEditorial.cNombre || ")" AS "Editorial"
FROM TLibro
     INNER JOIN TLibroAutor
           ON TLibro.nLibroID = TLibroAutor.nLibroID
     INNER JOIN TAutor
           ON TLibroAutor.nAutorID = TAutor.nAutorID
     INNER JOIN TAutorPais
           ON TAutor.nAutorID = TAutorPais.nAutorID
     INNER JOIN TPais
           ON TAutorPais.nPaisID = TPais.nPaisID
     INNER JOIN TEditorial
           ON TLibro.nEditorialID = TEditorial.nEditorialID
     INNER JOIN TPais AS TPaisEditorial
           ON TEditorial.nPaisID = TPaisEditorial.nPaisID;
```

Título	Autor	Editorial
Laura y Julio	Juan José Millás (España)	Alfaguara (España)
Rayuela	Julio Cortázar (Argentina)	Bruguera (España)
The Tempest	William Shakespeare (Gran Bretaña)	Penguin (Gran Bretaña)
...		

Figura 5.41
Consulta de libros, autores y editoriales, y salida por pantalla

Actividad propuesta 5.8

Partiendo de la base de datos de la figura 3.28, mostrar el nombre y país de todas las editoriales con los encabezados "Nombre" y "País", respectivamente.



5.1.8. Funciones integradas

Además de generar expresiones mediante los operadores aritméticos y de concatenación, los SGBD ofrecen una serie de funciones integradas en el gestor para facilitar el tratamiento de números, cadenas, fechas, etc. El catálogo de funciones integradas (*built-in functions*) de los SGBD comerciales suele ser bastante amplio. El estándar SQL define, entre otras, las siguientes:

CUADRO 5.4. Funciones integradas del estándar SQL

Función	Parámetros	Valor devuelto
ABS	Número	Valor absoluto del número
MOD	Número 1, Número 2	Módulo (resto) de la división del primer número entre el segundo
POSITION	Cadena de caracteres 1, Cadena de caracteres 2	Posición de la primera cadena dentro de la segunda
CHAR_LENGTH	Cadena de caracteres	Tamaño de la cadena
SUBSTRING	Cadena de caracteres, Posición de inicio, Número de caracteres	Cadena de caracteres dentro de la cadena recibida como parámetro. El segundo parámetro define la posición de la cadena original a partir de la que se extraerá la cadena que se va a devolver, y el tercero el número de caracteres de la cadena resultante
UPPER	Cadena de caracteres	Cadena de caracteres original en mayúsculas
LOWER	Cadena de caracteres	Cadena de caracteres original en minúsculas

```
SELECT ABS(SUM(nImporte))
FROM TFactura
WHERE dFactura BETWEEN "2013-01-01" AND "2013-12-31";
```

Figura 5.42. Uso de la función ABS()

TLibro			TEditorial		
nLibroID	cTitulo	nEditorial	nEditorial	cNombre	nPaisID
324	Clipper. Guía práctica	14	14	Anaya Multimedia	724


```

SELECT SUBSTRING(TLibro.cTitulo, 0, POSITION(TLibro.cTitulo, "Guía") - 1) AS "Titulo",
       UPPER(TEditorial.cNombre) AS "Editorial"
FROM TLibro
     INNER JOIN TEditorial
       ON TLibro.nEditorialID = TEditorial.nEditorialID
WHERE TLibro.nLibroID = 324;

```


Título	Editorial
Clipper.	ANAYA MULTIMEDIA

Figura 5.43. Uso de funciones de tratamiento de cadenas. Partiendo de los datos mostrados arriba, se selecciona la subcadena del campo TLibro.cTitulo desde el principio (posición cero) hasta la posición anterior a aquella donde aparece la palabra “Guía”. TEditorial.cNombre se convierte a mayúsculas

Actividad propuesta 5.9



Partiendo de la base de datos de la figura 3.28, mostrar en este orden el nombre, apellidos y NIF de todos los socios de forma que tras la concatenación del nombre, un espacio en blanco y los apellidos aparezca el número de caracteres de dicha concatenación, como se ve a continuación:

Socio	NIF
Paula Sanz González (19)	00789521T
José Luis García Viñals (23)	50687452Y
Javier Peinado Martín (21)	38546998X
...	

5.1.9. Inserción de registros

- *Campo a campo.* Se insertará un solo registro.

```

INSERT INTO <tabla> [( <campo1>[, <campo2>, ..., <campoN>])]
VALUES (<valor1>[, <valor2>, ..., <valorN>]);

```

Aunque la lista de campos es opcional, se recomienda especificarla siempre para que la asociación campo-valor no presente lugar a dudas.

```
INSERT INTO TSocio
    (cNIF, cNombre, cApellidos,
     cDireccion, cTelefono,
     dNacimiento, dAlta)
VALUES
    ("00789521T", "Paula", "Sanz González",
     "C/Martínez Izquierdo, 40", "619554687",
     "1983-09-13", "2013-10-03");
```

Figura 5.44. Inserción de un socio

Nótese que los valores van entrecomillados. Esto ocurre siempre con los campos de tipo cadena de caracteres, pero no siempre con los de tipo fecha, cuya especificación puede variar dependiendo del SGBD utilizado.

En la consulta del siguiente ejemplo no se ha incluido el campo nEditorialID (clave primaria) en la sentencia, ya que, al ser de tipo IDENTITY, su valor será asignado de forma automática por el SGBD. Por otro lado, el valor de nPaisID no va entrecomillado, al tratarse de un valor numérico.

```
INSERT INTO TEditorial (cNombre, nPaisID)
VALUES ("Alfaguara", 724);
```

Figura 5.45. Inserción de una editorial

- *Partiendo de una consulta.* Se insertarán tantos registros como devuelva la consulta.

```
INSERT INTO <tabla> [( <campo1>[, <campo2>, ..., <campoN>] ) ]
<consulta>;
```

```
INSERT INTO TPais
SELECT * FROM BDProveedores.TPais;
```

Figura 5.46
Relleno de TPais tomando la información de la tabla
de países de otra base de datos distinta

Actividad propuesta 5.10

Dar de alta un nuevo libro en la tabla TLibro de la base de datos de la figura 3.28.



5.1.10. Modificación de registros

La sentencia **UPDATE** permite actualizar los valores de campos concretos sobre los registros que cumplan una determinada condición. La condición puede estar compuesta por varios criterios relacionados mediante operadores lógicos. Si no se especifica la cláusula **WHERE** se actualizarán todos los registros de la tabla.

```
UPDATE <tabla>
  SET   <campo1> = <valor1>[,
        <campo2> = <valor2>[,
        ...
        <campoN> = <valorN>]
  [WHERE <condición>;
```

El siguiente ejemplo solo afecta a un registro, ya que la cláusula **WHERE** se refiere a un valor de clave primaria:

```
UPDATE TLibro
  SET nAnyoPublicacion = 1994
  WHERE nLibroID = 321;
```

Figura 5.47

Actualización del año de publicación de un libro concreto

En cambio, el ejemplo de la figura 5.48 puede afectar a varios autores, ya que se está utilizando un criterio menos restrictivo:

```
UPDATE TAutor
  SET cNombre = "Miguel Ángel"
  WHERE cNombre = "M. Ángel";
```

Figura 5.48. Actualización del nombre de varios autores

Como se apuntaba anteriormente, no especificar la cláusula **WHERE** afecta a todos los registros:

```
UPDATE TSocio
  SET dAlta = "2013-10-03";
```

Figura 5.49. Actualización de la fecha de alta de todos los socios

El ejemplo siguiente incluye un **WHERE** con condición múltiple:

```
UPDATE TLibro
  SET nAnyoPublicacion = 2010
  WHERE cTitulo = "El mundo"
        AND nEditorialID = 32;
```

Figura 5.50
Actualización del año de publicación de acuerdo con el título y la editorial

A continuación se actualizan varios campos a la vez:

```
UPDATE TSocio
  SET cDireccion = "C/Pez, 4",
      cTelefono = "677898899",
      dAlta = "2013-10-03"
  WHERE cNIF = "50687452Y";
```

Figura 5.51. Actualización de múltiples campos de un registro



Actividad propuesta 5.11

Modificar la tabla TLibro de la base de datos de la figura 3.28 para que el nuevo título del libro de código 312 sea "Historias de cronopios y de famas" y su año de edición 2007.

5.1.11. Eliminación de registros

La orden **DELETE** es análoga a la sentencia **UPDATE**. Si no se especifica la cláusula **WHERE**, se borrarán todos los registros de la tabla.

```
DELETE FROM <tabla>  
[WHERE <condición>];
```

```
DELETE FROM TSocio  
WHERE cNIF = "38546998X";
```

Figura 5.52. Borrado de un socio

```
DELETE FROM TLibro  
WHERE nEditorialID = 47;
```

Figura 5.53
Borrado de varios libros de acuerdo a un criterio (su código de editorial)

```
DELETE FROM TPais;
```

Figura 5.54. Borrado de todos los países

Actividad propuesta 5.12

En la base de datos de la figura 3.28, borrar todos los socios sin dirección dados de alta en 2014.



Se pueden utilizar subconsultas en operaciones de actualización y borrado, aunque algunos gestores no permiten modificar tablas que formen parte de la subconsulta:

```
UPDATE TEjemplar
  SET cSignatura = "689 LOP bas"
  WHERE nLibroID IN (SELECT nLibroID
                     FROM TLibro
                     WHERE nEditorialID = 32
                        AND nAnyoPublicacion = 2004);
```

Figura 5.55. Actualización basada en una subconsulta

5.2. Lenguaje de control de datos (DCL)

El lenguaje de control de datos (*data control language*, DCL) trabaja de forma activa con los permisos de seguridad que se conceden a los usuarios sobre los objetos de la base de datos (tablas, usuarios, tareas de administración, etc.). Asimismo, también sirve para gestionar transacciones, como se verá en breve.

5.2.1. Control de acceso a los datos

- *Concesión de privilegios.* Estos pueden ser concedidos sobre una tabla, una base de datos completa u otro objeto de la base de datos. Los permisos sobre tablas pueden ser de lectura (**SELECT**), inserción (**INSERT**), actualización (**UPDATE**) o borrado (**DELETE**). La opción **WITH GRANT OPTION** permite al usuario conceder a otro usuario los permisos que tiene a su alcance.

```
GRANT [CONNECT|RESOURCE|DBA|ALL PRIVILEGES|SELECT|UPDATE|INSERT|DELETE]
  ON <objeto>
  TO <lista_usuarios>
  [WITH GRANT OPTION];
```

```
GRANT ALL PRIVILEGES
  ON TSocio
  TO "GarciaF";
```

Figura 5.56

Se conceden todos los privilegios al usuario “GarciaF” sobre la tabla TSocio

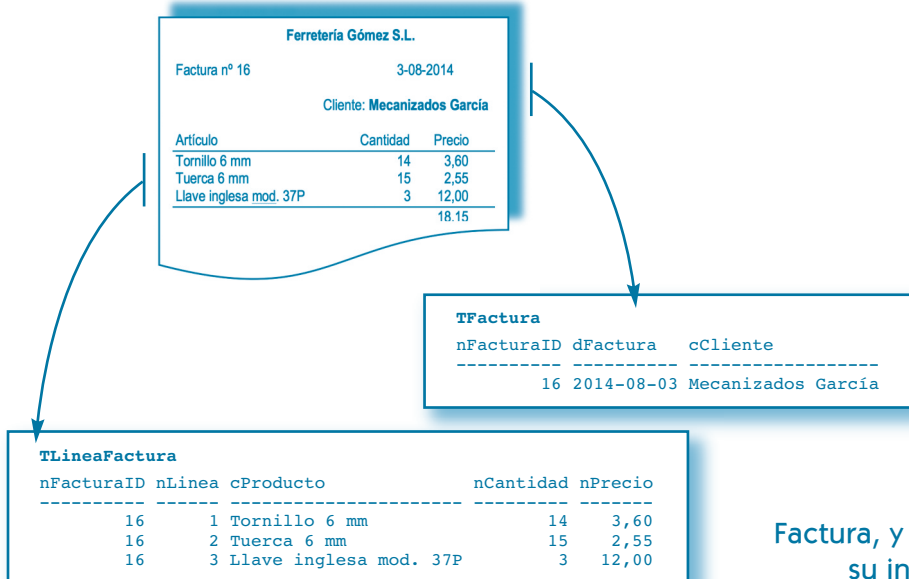
- *Revocación de privilegios.* Funciona de forma análoga a la sentencia **GRANT**, pero su efecto es el contrario.

```
REVOKE DELETE
FROM "GarciaF"
ON TSocio;
```

Se revocan los privilegios de borrado al usuario "GarciaF" sobre la tabla TSocio

5.2.2. Transacciones

Tomemos como ejemplo el alta de una factura. Esta operación implica escribir un nuevo registro en una tabla de facturas y un registro por cada línea que conforma el documento en una tabla de líneas de factura. Si damos de alta una factura con cuatro líneas y surge un error al escribir la tercera, nuestra base de datos quedará con información inconsistente (una factura compuesta por dos líneas cuyos importes no coincidirán con el total al ser sumados).



Factura, y almacenamiento de su información en tablas

La transacción es un mecanismo que contempla como un todo este tipo de acciones independientes pero relacionadas. Antes de iniciar las actualizaciones se abre la transacción. Si se ejecutan correctamente, quedan confirmadas y la transacción se cierra, pero si hay algún error la transacción se deshace por completo, garantizando la integridad de las inserciones, modificaciones y eliminaciones de datos.

El estándar SQL define una instrucción `START TRANSACTION` (`BEGIN TRANSACTION` en muchos SGBDR comerciales) que da por iniciada una transacción. Todas las sentencias de actualización posteriores no se harán efectivas hasta que se encuentre una sentencia de confirmación (`COMMIT`). Si se detectase algún error durante la transacción, se puede deshacer volviendo al estado original mediante la sentencia `ROLLBACK`.

```
START TRANSACTION

INSERT INTO TFactura ...

INSERT INTO TAlbaran ...

INSERT INTO TAlbaran ...

...

INSERT INTO TAlbaran ...

IF ERROR
    ROLLBACK
ELSE
    COMMIT
```

Figura 5.59. Ejemplo habitual de transacción (código no estándar)

Toda transacción debe cumplir las normas ACID:

- *Atomicity* (atomicidad). La transacción debe funcionar como un proceso atómico. O tienen éxito todas las sentencias procesadas en una transacción o no lo tiene ninguna.
- *Consistency* (consistencia). La base de datos quedará en un estado consistente tras la ejecución de la transacción.
- *Isolation* (aislamiento). Cada transacción debe ejecutarse con independencia del resto de las transacciones. Es difícil garantizar esta norma, ya que su aplicación estricta contradice normas básicas de eficiencia empresarial. En una base de datos con mucho trasiego no se puede esperar a que acabe cada transacción para iniciar una nueva.
- *Durability* (persistencia). Los cambios efectuados tras la ejecución de una transacción deben permanecer en la base de datos aunque haya errores de *hardware* posteriores.

Las transacciones son delicadas de gestionar, ya que presentan problemas relacionados con los accesos concurrentes vistos en el tema 3. Uno de ellos surge cuando dos transacciones que se ejecutan de forma simultánea acceden al mismo dato. En ese caso, surgen tres posibles problemas:

- *Dirty read* (lectura sucia). Una transacción A consulta información escrita por otra transacción B que aún no ha sido confirmada. Si la transacción B acabase en un **ROLLBACK**, la transacción A estaría manejando información incorrecta.
- *Nonrepeatable read* (lectura irrepetible). Una transacción A consulta un valor dos veces y obtiene distinto resultado, ya que dicho dato ha sido modificado por una transacción B en el intervalo entre ambas consultas.
- *Phantom read* (lectura fantasma). Al efectuar una consulta, una transacción A encuentra unos datos que no existían cuando se inició. Otra transacción B los ha insertado.

Como se vio anteriormente, la solución a este tipo de problemas pasa por implementar políticas de bloqueo, pero estas, a su vez, causan otros problemas. El más habitual es el *deadlock*, situación en la que dos transacciones quedan bloqueadas indefinidamente, ya que ambas intentan acceder a datos bloqueados por la otra transacción. En ese caso, el SGBDR debe implementar algoritmos de detección de *deadlocks*, forzando el **ROLLBACK** de una de las transacciones comprometidas.

PARA SABER MÁS



Una antigua ley del Estado de Texas establecía que “cuando dos trenes se acerquen al mismo cruce, ambos deben parar por completo y ninguno debe volver a arrancar hasta que el otro se haya ido”. He ahí un ejemplo de *deadlock*.

Resumen

El DML es la parte del lenguaje SQL que se encarga de la gestión de la información contenida en una base de datos, y la sentencia **SELECT** es su herramienta de recuperación de información. Su sintaxis varía entre la sencillez de las consultas a una tabla y la complejidad de las composiciones internas y externas (consultas a varias tablas, en el segundo de los casos incluyendo los registros que no guardan correspondencia con registros en otras tablas). **SELECT** también permite embeber consultas (subconsultas) dentro de consultas y combinar consultas mediante los operadores **UNION**, **MINUS** e **INTERSECT**. La información que se obtiene se depura aún más gracias al uso de funciones de agregación usadas en solitario o con las cláusulas **GROUP BY** y **HAVING**.

Además de gestionar las consultas a la base de datos, el DML también se encarga de las inserciones, actualizaciones y borrados, gracias a las sentencias **INSERT**, **UPDATE** y **DELETE**, respectivamente.

Otra parte de SQL es el DCL, encargado de las operaciones de control. Mediante el DCL se pueden conceder o revocar permisos a usuarios y gestionar transacciones, es decir, secuencias de operaciones relacionadas que se tratan como una sola operación a efectos de la información resultante en la base de datos.

EJERCICIOS PROPUESTOS

1. Sobre la base de datos de la figura 3.28 escribir las sentencias **SELECT** que obtengan los siguientes resultados:
 - a) Los títulos de los libros publicados entre 1926 y 1978 que no estén publicados por la editorial número 32.
 - b) El nombre y apellidos de todos los socios dados de alta después de 2009 que cuenten con dirección.
 - c) Todos los códigos de país en los que haya editoriales, sin valores repetidos.
 2. Sobre la base de datos de la figura 3.28, escribir las sentencias **SELECT** que obtengan los siguientes resultados:
 - a) Los títulos de los libros cuyo nombre comience por “Guía” que no estén publicados por la editorial “Alfaguara”.
 - b) La lista de temas para los que la editorial “Alfaguara” tiene libros publicados, sin valores repetidos.
 - c) Los títulos de los libros que nunca han sido tomados en préstamo.
 3. Sobre la base de datos de la figura 3.28, escribir las sentencias **SELECT** que obtengan los siguientes resultados:
 - a) El número de libros existentes por editorial (mostrar el nombre de la editorial y el número de libros, este último bajo el encabezado “Número de libros”).
 - b) El número de socios que se han llevado algún libro en préstamo durante el año 2013.
 - c) El número de autores de todos los libros con más de dos autores (mostrar el título del libro y el número de autores, este último bajo el encabezado “Número de autores”).
 4. Sobre la base de datos de la figura 3.28, escribir sentencias de inserción de registros en las tablas TTema y TPrestamo.
 5. Sobre la base de datos de la figura 3.28, escribir la sentencia que asigne “01/08/2014” como fecha de alta de los socios que no cuentan con dirección.
-

ACTIVIDADES DE AUTOEVALUACIÓN

1. El DML permite:
 - ☐ a) Crear, modificar y borrar objetos de la base de datos.
 - ☐ b) Dar de alta, modificar, borrar y consultar información de las tablas.
 - ☐ c) Gestionar permisos de usuarios y transacciones.
2. Para comprobar si un campo tiene valor:
 - ☐ a) Hay que compararlo con una cadena vacía o con el número 0, dependiendo de su tipo de datos.
 - ☐ b) Hay que compararlo con el valor `NULL`.
 - ☐ c) Hay que aplicarle el operador `IS NULL`.
3. La cláusula `WHERE` que permite comprobar si el campo `cNombre` empieza por "Mart" es:
 - ☐ a) `WHERE cNombre LIKE "Mart"`.
 - ☐ b) `WHERE cNombre LIKE "Mart_"`.
 - ☐ c) `WHERE cNombre LIKE "Mart%"`.
4. La cláusula `HAVING`:
 - ☐ a) Puede aparecer en una consulta independientemente de `GROUP BY`.
 - ☐ b) Se puede aplicar sobre cualquier campo o expresión.
 - ☐ c) Solo se puede aplicar sobre funciones de agregación.
5. El *deadlock* se produce:
 - ☐ a) Cuando dos transacciones quieren acceder a la vez al mismo dato.
 - ☐ b) Cuando dos transacciones quieren acceder a datos bloqueados por la otra transacción.
 - ☐ c) Como consecuencia del *dirty read*.

SOLUCIONES:

1. ☐ a ☒ b ☐ c2. ☐ a ☐ b ☒ c3. ☐ a ☐ b ☒ c4. ☐ a ☐ b ☒ c5. ☐ a ☒ b ☐ c

Lenguaje de programación. Construcción de guiones

Objetivos

1. Comprender por qué es necesario contar con un lenguaje de programación Turing completo en el SGBD.
2. Entender la utilidad de los guiones de administración.
3. Manejar correctamente la sintaxis de SQL/PSM.
4. Dominar el uso de cursores para gestionar los resultados de una consulta.
5. Sacar partido a las posibilidades que ofrecen las funciones y procedimientos almacenados.
6. Gestionar excepciones con corrección y precisión.
7. Entender el funcionamiento de los *triggers*.

Mapa conceptual del capítulo



Glosario

Código fuente. Instrucciones escritas en un lenguaje de programación, generalmente en un fichero de texto.

Invocar. En contexto de programación, ejecutar una función o procedimiento desde otra parte del código.

Lenguaje de cuarta generación (4GL). Lenguaje de programación que refina las características de los lenguajes de programación tradicionales, permitiendo al usuario conseguir sus objetivos sin necesidad de trabajar a un gran nivel de detalle.

Palabra reservada. Palabra que desempeña una funcionalidad en un lenguaje de programación, motivo por el que el programador no puede utilizarla para dar nombre a una variable u otro objeto.

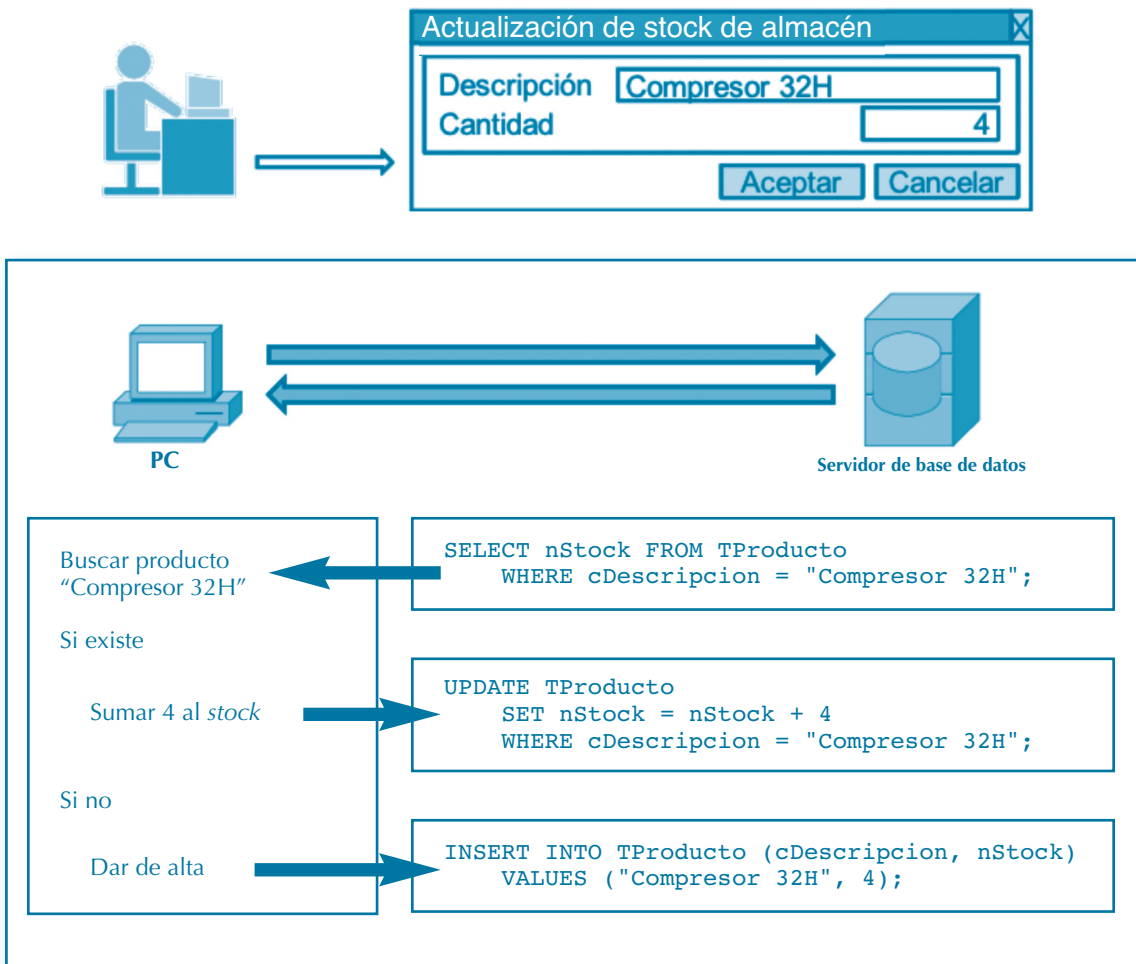
Puntero. Elemento que expresa la posición de memoria de un valor (“apunta” a dicho valor).

Servidor de base de datos. Ordenador que almacena bases de datos de una organización, así como el *software* que permite acceder a ellas (el SGBD).

6.1. La necesidad de un lenguaje de programación en el SGBD

En los dos temas anteriores se ha abordado el lenguaje SQL y la ejecución de sus instrucciones de forma aislada. No obstante, lo habitual en cualquier *software* que trabaje con una base de datos es ejecutar secuencias de instrucciones que realicen varias comprobaciones, tareas secuenciales o repetitivas, etc. El código fuente del lenguaje de programación de dicho *software* debe enviar constantemente sentencias SQL al servidor de base de datos.

En el siguiente ejemplo, un usuario introduce información sobre un producto en una aplicación de gestión de *stock* de un almacén. La aplicación se ejecuta en un PC, pero debe acceder a la información almacenada en una base de datos residente en un servidor:

**Figura 6.1**

Proceso de actualización de *stock* en la base de datos. La aplicación *software* que se ejecuta en el PC debe efectuar tres llamadas independientes al servidor para ejecutar las tres sentencias SQL

La ejecución de sentencias SQL de forma independiente implica una comunicación continua entre la aplicación y la base de datos, con el consiguiente tráfico de red. Por ese motivo, ISO definió el estándar SQL/PSM (*SQL/persistent stored modules*, SQL/módulos almacenados persistentes), integrado en el estándar ANSI/SQL desde 1999. SQL/PSM añade a las funcionalidades de SQL el uso de estructuras de control y declaración de variables, constantes y funciones, conformando un lenguaje de cuarta generación (4GL) capaz de combinar la capacidad de un lenguaje de programación Turing completo con la facilidad de programación de los lenguajes no procedimentales. SQL/PSM permite almacenar y ejecutar bloques de código fuente en el servidor de base de datos, disminuyendo el trasiego de información por la red e independizando el tratamiento de los datos del resto de funciones del *software* de gestión, en línea con las arquitecturas de desarrollo en capas imperantes en los últimos años:

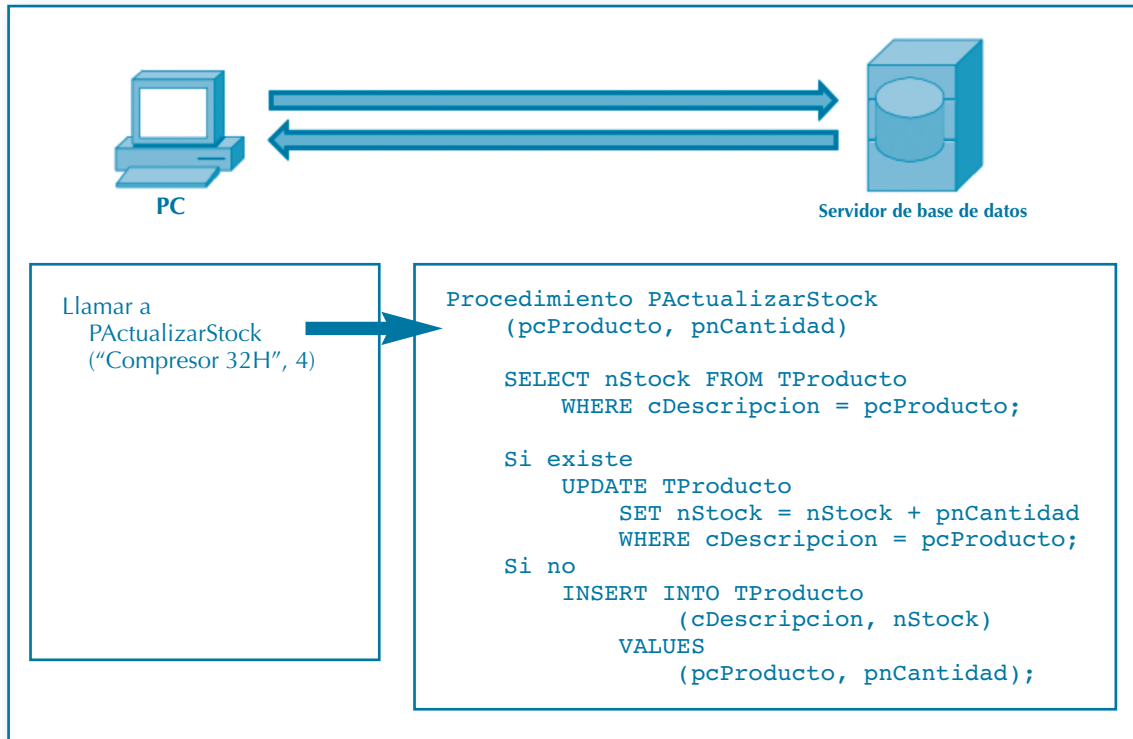
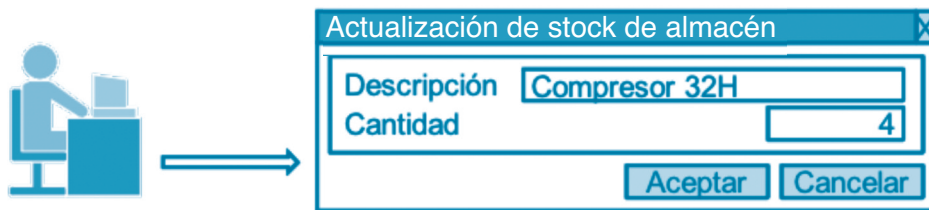


Figura 6.2
Proceso de actualización de stock en la base de datos
con SQL/PSM (código no estándar)

Actividad propuesta 6.1

La comunicación descrita en las figuras anteriores entre el *software* presente en un PC y la base de datos radicada en un servidor obedece a la arquitectura informática conocida como modelo cliente/servidor. Buscar información sobre dicho modelo y sobre los servicios que ofrecen las bases de datos relacionales.



6.1.1. Construcción de guiones

El ejemplo de la figura 6.2 solo representa una de las utilidades de SQL/PSM. Disponer de un lenguaje estructurado integrado en el SGBD también permite desarrollar guiones de administración, es decir, bloques de código dedicados a tareas de gestión y mantenimiento de la base de datos:

- Creación, borrado o actualización masiva de permisos de usuario.
- Generación/comprobación de estadísticas de uso de la base de datos.
- *Backups*.
- Transacciones.

Los SGBD comerciales suelen ofrecer la capacidad de temporizar bloques de código SQL/PSM, es decir, programar su ejecución en un instante de tiempo concreto con o sin periodicidad. De ese modo, el administrador de la base de datos puede establecer la ejecución de *backups* a ciertas horas de la madrugada (aprovechando que no habrá usuarios trabajando con las bases de datos) o bien puede programar la generación de estadísticas a primera hora de la mañana, justo antes de que el personal de bases de datos llegue a la oficina.

6.2. Programación en SQL/PSM

Las implementaciones comerciales de SQL/PSM más populares son el lenguaje PL-SQL de Oracle y el Transact-SQL de Microsoft (integrado en su gestor Microsoft SQL Server), si bien las versiones más cercanas al estándar son las de MySQL y DB2. En este tema se verá la sintaxis del estándar SQL:1999. Se invita al lector a adaptar el código fuente SQL/PSM de los ejemplos al del SGBD de su elección.

En este apartado se repasarán los distintos elementos de la sintaxis de programación SQL/PSM. Aunque no es estrictamente necesario, se recomienda contar con conocimientos básicos de programación estructurada.



Actividad propuesta 6.2

Elaborar una lista con los lenguajes procedimentales de los SGBD(O)R más populares. Se puede partir de la relación ofrecida en el cuadro 1.1 del capítulo 1.

6.2.1. Variables. Cursores

Las variables son contenedores de memoria donde el programador puede almacenar valores de forma temporal. Cada variable debe declararse mediante la palabra reservada **DECLARE** indicando el tipo de datos de los valores que puede albergar.

```
DECLARE <variable> <tipo>;
```

También se pueden declarar varias variables a la vez, si todas corresponden al mismo tipo de datos:

```
DECLARE <variable1>, <variable2>, ..., <variableN> <tipo>;
```

A lo largo de un programa a una variable se le puede asignar un valor o expresión tantas veces como se desee. Para asignar un valor a una variable se utiliza la palabra **SET**:

```
SET <variable> = <valor>;
```

```
DECLARE vcTexto VARCHAR(100);
DECLARE vnImporte, vnIVA FLOAT;

...
SET vnImporte = 1430;
...
SET vnIVA = 21;
SET vnImporte = vnImporte + ((vnImporte / 100) * vnIVA)
SET vcTexto = "Importe final: " || vnImporte || " €.";
...
...
```

Figura 6.3. Declaración y asignación de variables

Nótese que, como ampliación de la Notación Húngara comentada en el tema 3, y para diferenciar variables de nombres de campo, en el presente texto los nombres de variable irán precedidos por una “v” minúscula.

SQL/PSM cuenta con variables de sistema, es decir, valores a los que el programador puede acudir cuando lo desee. Algunos de los más útiles son los siguientes:

CUADRO 6.1. Algunas variables de sistema de SQL/PSM

CURRENT_DATE	Fecha actual
CURRENT_TIME	Hora actual
CURRENT_TIMESTAMP	Fecha y hora actual, usada para funciones de auditoría
CURRENT_USER	Usuario que está conectado actualmente al SGBD
SQLSTATE	Estado actual, incluyendo códigos de error

En algunos SGBD los valores del cuadro no están disponibles en variables de sistema, sino en forma de funciones integradas.

Mediante las variables se pueden recuperar datos de una consulta SQL, tarea que se puede realizar de tres formas distintas:

- a) Si la consulta devuelve un único valor, mediante la palabra **SET**:

```
SET <variable> = <consulta>;
```

```
DECLARE vcNombreSocio VARCHAR(91);

SET vcNombreSocio = SELECT cNombre || " " || cApellidos
                     FROM TSocio
                     WHERE cNIF = "50687452Y";

...
```

Figura 6.4. Asignación del resultado de una consulta a una variable

- b) Si la consulta devuelve un valor, mediante la cláusula **INTO**:

```
SELECT <campo>
      INTO <variable>
      [<resto_consulta>];
```

```
DECLARE vcNombreSocio VARCHAR(91);

SELECT cNombre || " " || cApellidos
      INTO vcNombreSocio
      FROM TSocio
      WHERE cNIF = "50687452Y";

...
```

Figura 6.5. Asignación del resultado de una consulta a una variable

- c) Independientemente del número de campos y registros que devuelva la consulta, mediante el uso de un *cursor*. Los cursores son estructuras de almacenamiento volátil que albergan el resultado de una consulta y permiten recorrer todos sus registros. Para utilizar un cursor hay que seguir los siguientes pasos:

1. Declarar el cursor. Si se especifica la cláusula **SCROLL** se está indicando que el cursor no se recorrerá secuencialmente (desde el primer elemento en adelante), sino que es posible que se recorra hacia atrás o a saltos.

```
DECLARE <nombre_cursor> [SCROLL] CURSOR
      FOR <consulta>;
```

2. Abrir el cursor.

```
OPEN <nombre_cursor>;
```

3. Recorrer el cursor. Cada vez que se ejecuta la sentencia **FETCH** se lee un nuevo registro. Si la consulta devuelve varios campos sus valores se pueden almacenar en varias variables, separadas por comas.

```
FETCH [NEXT|PRIOR|FIRST|LAST|RELATIVE <núm_registros>|ABSOLUTE
<núm_registros>]
FROM <nombre_cursor>
INTO <variable1>[, <variable2>, ..., <variableN>];
```

Si se especifica el valor **NEXT**, la sentencia **FETCH** devolverá los valores del siguiente registro disponible. Dado que **NEXT** es el valor por defecto, no es necesario especificarlo.

Si se especifica **PRIOR**, se devolverán los datos del registro anterior. **FIRST** se desplaza hasta el primer registro y **LAST** hasta el último. **RELATIVE** se desplaza desde el registro actual tantos registros como indique **<núm_registros>**. Si **<núm_registros>** es positivo se desplazará hacia adelante, si es negativo hacia atrás. **ABSOLUTE** se posicionará en el número de registro especificado por **<núm_registros>** si este es positivo, o bien hacia atrás desde el último registro si **<núm_registros>** es negativo. Para utilizar cualquier cláusula distinta de **NEXT** es necesario especificar **SCROLL** en la declaración del cursor.

4. Cerrar el cursor.

```
CLOSE <nombre_cursor>;
```

```
DECLARE vcNombreSocio VARCHAR(30);
DECLARE vcApellidosSocio VARCHAR(60);

DECLARE curSocios CURSOR
FOR SELECT TSocio.cNombre, TSocio.cApellidos
FROM TSocio
INNER JOIN TPrestamo
ON TSocio.cNIF = TPrestamo.cNIF
WHERE TPrestamo.dPrestamo BETWEEN "2014-07-01" AND "2014-08-31";
...

OPEN curSocios;

FETCH FROM curSocios INTO vcNombreSocio, vcApellidosSocio;
...

FETCH FROM curSocios INTO vcNombreSocio, vcApellidosSocio;
...

CLOSE curSocios;
```

Figura 6.6. Ejemplo de uso de un cursor. Se leen los dos primeros registros

Nuevamente, para adaptarnos a la notación húngara, y al tratarse los cursores de tipos especiales de variables, los prefijaremos con las letras “cur” en minúscula.

En SQL/PSM los cursores permiten modificar o borrar el registro en el que se esté posicionado. Para ello hay que ejecutar las siguientes versiones de las sentencias UPDATE y DELETE:

```
UPDATE <nombre_tabla>
  SET <campo1> = <valor1>[,
    <campo2> = <valor2>[,
    ...
    <campoN> = <valorN>]
  WHERE CURRENT OF <nombre_cursor>;

DELETE <nombre_tabla>
  WHERE CURRENT OF <nombre_cursor>;
```

```
DECLARE vcSignatura VARCHAR(15);
DECLARE vcNIF CHAR(9);
DECLARE vdPrestamo DATE;

DECLARE curPrestamos CURSOR
  FOR SELECT cSignatura, cNIF, dPrestamo
     FROM TPrestamo;
...

OPEN curPrestamos;

FETCH FROM curPrestamos INTO vcSignatura, vcNIF, vdPrestamo;
IF vdPrestamo >= "3000-01-01" THEN
  DELETE FROM TPrestamo
    WHERE CURRENT OF curPrestamos;
END IF;
...

FETCH FROM curSocios INTO vcNombreSocio, vcApellidosSocio;
...

CLOSE curPrestamos;
```

Figura 6.7. Ejemplo de borrado dentro de un cursor. Se buscan libros con fechas incorrectas y se borran en el momento



Actividad propuesta 6.3

Partiendo de la base de datos de la figura 3.28, escribir código SQL/PSM que calcule el número de libros editado en el año 1992. Posteriormente habrá que almacenar en una cadena de caracteres el texto “En el año 1992 se editaron <n1> libros.”, donde <n1> es el número de libros editados en 1992.

6.2.2. Estructuras de control de flujo

En todo lenguaje de programación las estructuras de control son las que permiten que el programa resultante tome decisiones, evalúe condiciones, repita tareas y, en definitiva, defina el flujo del código, es decir, el orden en que se ejecutarán las distintas sentencias. SQL/PSM contempla las estructuras de control de flujo habituales, a saber:

- **IF.** Evalúa una condición, llevando a cabo una acción en caso de que la condición sea cierta.

```
IF <condición> THEN
    <lista_sentencias>
[ELSEIF <condición> THEN
    <lista_sentencias>]
[ELSE
    <lista_sentencias>]
END IF;
```

```
DECLARE vnImporte FLOAT;

SET vnImporte = SELECT nImporte
                 FROM TFactura
                 WHERE nFacturaID = 37;

IF vnImporte = 0 THEN
    DELETE FROM TFactura
           WHERE nFacturaID = 37;
END IF;
...
```

Figura 6.8
Ejemplo de IF. Se comprueba el importe de una factura y, si es cero, se elimina

Opcionalmente se puede utilizar la palabra **ELSE** para llevar a cabo otra acción si la condición es falsa.

```
DECLARE vnImporte FLOAT;

SET vnImporte = SELECT nImporte
                 FROM TFactura
                 WHERE nFacturaID = 37;

IF vnImporte = 0 THEN
    DELETE FROM TFactura
           WHERE nFacturaID = 37;
ELSE
    UPDATE TFactura
           SET nImporte = nImporte + 100
           WHERE nFacturaID = 37;
END IF;
...
```

Figura 6.9
Ejemplo de IF con ELSE. Se comprueba el importe de una factura. Si es cero, se elimina, pero si es distinto de cero se incrementa en 100

SQL/PSM, al igual que otros lenguajes, también incluye la palabra **ELSEIF** para evitar la anidación excesiva de sentencias **IF**.

```
DECLARE vnImporte FLOAT;

SET vnImporte = SELECT nImporte
                FROM TFactura
                WHERE nFacturaID = 37;

IF vnImporte = 0 THEN
    DELETE FROM TFactura
        WHERE nFacturaID = 37;
ELSE
    IF vnImporte < 0 THEN
        UPDATE TFactura
            SET nImporte = 50
            WHERE nFacturaID = 37;
    ELSE
        IF vnImporte < 100 THEN
            UPDATE TFactura
                SET nImporte = nImporte + 50
                WHERE nFacturaID = 37;
        ELSE
            UPDATE TFactura
                SET nImporte = nImporte + 100
                WHERE nFacturaID = 37;
        END IF;
    END IF;
END IF;
...
```

```
DECLARE vnImporte FLOAT;

SET vnImporte = SELECT nImporte
                FROM TFactura
                WHERE nFacturaID = 37;

IF vnImporte = 0 THEN
    DELETE FROM TFactura
        WHERE nFacturaID = 37;
ELSEIF vnImporte < 0 THEN
    UPDATE TFactura
        SET nImporte = 50
        WHERE nFacturaID = 37;
ELSEIF vnImporte < 100 THEN
    UPDATE TFactura
        SET nImporte = nImporte + 50
        WHERE nFacturaID = 37;
ELSE
    UPDATE TFactura
        SET nImporte = nImporte + 100
        WHERE nFacturaID = 37;
END IF;
...
```

Figura 6.10. Ejemplo de **IF** con **ELSEIF**. Ambos bloques de código son equivalentes, pero el de abajo evita la anidación de sentencias **IF**, lo que favorece la legibilidad del código fuente

Actividad propuesta 6.4



Declarar una variable `vcTexto` de tipo cadena de caracteres. En la base de datos de la figura 3.28, comprobar si el libro de código 342 tiene uno o varios autores. En el primer caso, asignar a la variable el texto "Tiene un autor"; en el segundo, asignar "Tiene varios autores".

- **CASE.** Compara el resultado de una expresión con varios valores, llevando a cabo una acción concreta si la expresión se evalúa a verdadero para alguno de ellos. De forma análoga a **IF**, también incluye una cláusula **ELSE** si la expresión no se evalúa a verdadero al compararla con ninguno de los valores propuestos.

```
CASE <expresión>
  WHEN <valor1> THEN
    <lista_sentencias>
  [WHEN <valor2> THEN
    <lista_sentencias>
    ...
  WHEN <valorN> THEN
    <lista_sentencias>]
  [ELSE
    <lista_sentencias>]
END CASE;
```

```
DECLARE vcPais VARCHAR(30);
DECLARE vcCodPais VARCHAR(3);

SET vcPais = SELECT DISTINCT TPais.cNombre
              FROM TPais
              INNER JOIN TEditorial
                ON TPais.nPaisID = TEditorial.nPaisID
              WHERE TEditorial.nEditorialID = 32;

CASE vcPais
  WHEN "España" THEN
    SET vcCodPais = "ESP";
  WHEN "Estados Unidos" THEN
    SET vcCodPais = "USA";
  WHEN "Francia" THEN
    SET vcCodPais = "FRA";
  ELSE
    SET vcCodPais = "";
END CASE;
...
```

Figura 6.11. Ejemplo de CASE. Se rellena una variable con un código alfanumérico de país correspondiente al país de una editorial concreta



Actividad propuesta 6.5

Modificar el código de la actividad propuesta 6.4 para que a la variable `vcTexto` se le asigne el valor "Tiene un autor", "Tiene dos autores", "Tiene tres autores", "Tiene cuatro autores" y "Tiene más de cuatro autores" si el libro de código 342 tiene uno, dos, tres, cuatro o más de cuatro autores, respectivamente.

A partir de este momento se verán las *sentencias iterativas* o *repetitivas*, llamadas de ese modo porque permiten repetir una o varias líneas de código hasta que se cumpla una condición determinada. Al bloque de código delimitado por una sentencia iterativa se lo conoce como *bucle*.

- **WHILE.** Evalúa al principio del bucle la condición que ha de cumplirse para seguir iterando dentro del mismo. En cuanto dicha condición no se cumpla, el flujo de código continuará en la sentencia inmediatamente siguiente al final del bucle. Si la condición no se cumple la primera vez que el flujo del código llega al bucle, este no se ejecuta nunca.

```
WHILE <condición_de_permanencia> DO
    <lista_sentencias>
END WHILE;
```

```
DECLARE vnFacturaID INT;
DECLARE vnImporte, vnImporteAcumulado FLOAT;
DECLARE vcTexto VARCHAR(100);

DECLARE curFacturas CURSOR
FOR SELECT nFacturaID, nImporte
FROM TFactura
WHERE dFactura BETWEEN "2013-01-01" AND "2013-12-31"
ORDER BY dFactura;

...

SET vnImporteAcumulado = 0;

OPEN curFacturas;

WHILE vnImporteAcumulado < 1000000 DO
    FETCH FROM curFacturas INTO vnFacturaID, vnImporte;

    SET vnImporteAcumulado = vnImporteAcumulado + vnImporte;
END WHILE;

SET vcTexto = "La factura un millón fue la " || vnFacturaID;

CLOSE curFacturas;

...
```

Figura 6.12. Ejemplo de WHILE. Recorre un cursor con las facturas del año 2013 ordenadas cronológicamente hasta que encuentra la que permitió a la empresa superar el millón de euros de facturación

Actividad propuesta 6.6



Escribir un bloque de código que realice la siguiente tarea a partir de la base de datos de la figura 3.28:

1. Crear una tabla de nombre TPrestamosAntiguos con los campos cSignatura (VARCHAR(15)) y dPrestamo (DATE).
2. Crear un cursor que consulte todos los datos de TPrestamo ordenados por fecha de préstamo de forma ascendente (los más antiguos primero).
3. Consultar en un bucle WHILE cada registro devuelto por el cursor hasta que la fecha de préstamo sea posterior al año 1999. Por cada registro se insertará un nuevo registro en TPrestamosAntiguos con la signatura y la fecha de préstamo consultadas.

- **REPEAT.** La evaluación se produce al final del bucle, y la condición no es de permanencia, sino de salida. Es decir, el bucle **REPEAT** ejecutará su contenido hasta que se cumpla la condición especificada en la sentencia **UNTIL**, por lo que se ejecutará, como mínimo, una vez.

```
REPEAT
    <lista_sentencias>
UNTIL <condición_de_salida>
END REPEAT;
```

```
DECLARE vnFacturaID INT;
DECLARE vnImporte, vnImporteAcumulado FLOAT;
DECLARE vcTexto VARCHAR(100);

DECLARE curFacturas CURSOR
    FOR SELECT nFacturaID, nImporte
        FROM TFactura
        WHERE dFactura BETWEEN "2013-01-01" AND "2013-12-31"
        ORDER BY dFactura;
...

SET vnImporteAcumulado = 0;

OPEN curFacturas;

REPEAT
    FETCH FROM curFacturas INTO vnFacturaID, vnImporte;

    SET vnImporteAcumulado = vnImporteAcumulado + vnImporte;
UNTIL vnImporteAcumulado >= 1000000
END REPEAT;

SET vcTexto = "La factura un millón fue la " || vnFacturaID;

CLOSE curFacturas;
...
```

Figura 6.13. Ejemplo de REPEAT. Lleva a cabo el mismo proceso que la figura 6.12, pero la condición es opuesta, al ser condición de salida y no de permanencia



Actividad propuesta 6.7

Reescribir el código de la actividad propuesta 6.6 con un bucle REPEAT en vez de WHILE.

- **FOR.** El bucle FOR difiere considerablemente de su aplicación habitual en la mayoría de los lenguajes de programación. En SQL/PSM está especialmente diseñado para recorrer un cursor, cuya declaración, apertura y cierre lleva a cabo de forma implícita. Es, por tanto, la estructura iterativa recomendada para recorrer cursores por completo.

```
FOR <nombre_for> AS <nombre_cursor> CURSOR
  FOR <consulta>
  DO
    <lista_sentencias>
END FOR;
```

```
DECLARE vcNIF CHAR(9);
DECLARE vcNombre VARCHAR(30);
DECLARE vcApellidos VARCHAR(60);
DECLARE vdAlta DATE;

...

CREATE TABLE TSocioNuevo (
  cNIF CHAR(9) NOT NULL UNIQUE PRIMARY KEY,
  cNombre VARCHAR(30) NOT NULL,
  cApellidos VARCHAR(60) NOT NULL
);

ALTER TABLE TSocio
  ADD lVigente BOOLEAN DEFAULT TRUE;

FOR forSocios AS curSocios CURSOR
  FOR SELECT TSocio.cNIF, TSocio.cNombre, TSocio.cApellidos, TSocio.dAlta
  FROM TSocio;
DO
  FETCH FROM curSocios INTO vcNIF, vcNombre, vcApellidos, vdAlta;

  IF vdAlta >= "2010-01-01" THEN
    INSERT INTO TSocioNuevo
      (cNIF, cNombre, cApellidos)
    VALUES
      (vcNIF, vcNombre, vcApellidos);
  ELSE
    IF NOT EXISTS(SELECT *
                  FROM TPrestamo
                  WHERE cNIF = vcNIF)
      UPDATE TSocio
        SET lVigente = FALSE
        WHERE cNIF = vcNIF;
    END IF;
  END IF;
END FOR;
...
```

Figura 6.14. Ejemplo de FOR. Incluye un nuevo campo lógico lVigente en TSocio, a cierto por defecto. Si el socio se dio de alta antes del año 2010 y no se ha llevado ningún libro en préstamo, se actualizará a falso el valor de lVigente. Si el socio se dio de alta en 2010 o posteriormente, sus datos se darán de alta en la nueva tabla TSocioNuevo. Nótese el uso de la función EXISTS() para comprobar si el socio ha efectuado préstamos

Actividad propuesta 6.8



Partiendo de la base de datos de la figura 3.28, escribir código SQL/PSM que lleve a cabo los siguientes pasos:

1. Crear una tabla TSocioAntiguo con los campos cNombre (VARCHAR(30)) y cApellidos (VARCHAR(60)).
2. Escribir un bucle FOR que lea un cursor con los datos de todos los socios.
3. En cada iteración del bucle, comprobar si la fecha de nacimiento del socio es anterior a 1940. En caso afirmativo, insertar en TSocioAntiguo un nuevo registro con el nombre y el apellido de dicho socio.

SQL/PSM provee otro tipo de bucle, la estructura `LOOP...END LOOP`, cuya salida ha de forzarse mediante la instrucción `LEAVE`. Dado que esa ruptura del flujo contradice las normas básicas de la programación estructurada, además de que no es necesario utilizarla, no se verá en este libro.

6.2.3. Subrutinas. Procedimientos y funciones almacenados

Hasta ahora se ha hablado de pequeños elementos dentro del código fuente SQL/PSM. Ha llegado el momento de agruparlos. Todo bloque de código (a excepción de las declaraciones de variables) debe ir entre un par de instrucciones `BEGIN` y `END`, como se muestra en el siguiente ejemplo:

```
DECLARE vnImporte FLOAT;

BEGIN
    SET vnImporte = SELECT nImporte
                    FROM TFactura
                    WHERE nFacturaID = 37;

    IF vnImporte = 0 THEN
        DELETE FROM TFactura
            WHERE nFacturaID = 37;
    ELSE
        UPDATE TFactura
            SET nImporte = nImporte + 100
            WHERE nFacturaID = 37;
    END IF;
END;
```

Figura 6.15. Código de la figura 6.9 delimitado por `BEGIN` y `END`

Una vez que queda claro cómo escribir código, el siguiente paso es almacenar los bloques de código en el SGBD para poder ejecutarlos cuando el programador lo desee. Dichos bloques se guardan en forma de *subrutinas*, a saber, *procedimientos almacenados* y *funciones almacenadas*. Los procedimientos, cuyo nombre prefijaremos con una “P” (mayúscula), tan solo ejecutan bloques de código, mientras las funciones devuelven un valor como resultado de sus cálculos. Por ese motivo prefijaremos su nombre con una “F” (mayúscula) y la inicial del tipo de datos del valor que devuelven. Ambos pueden recibir valores a partir de los cuales llevar a cabo sus funcionalidades. Dichos valores se llaman *parámetros*, y los identificaremos precediendo su nombre de una “p” (minúscula).

Al almacenar una subrutina, esta puede ser invocada desde otra subrutina, desde el código fuente de un lenguaje de programación externo o desde la interfaz de línea de comandos del SGBD.

- Los procedimientos tienen la siguiente sintaxis:

```
CREATE PROCEDURE <nombre_procedimiento> ([<par1>[, <par2>,...,<parN>]])
<declaraciones>
BEGIN
    <lista_sentencias>
END;
```

Cada parámetro se especifica del siguiente modo:

```
[ IN | OUT | INOUT ] <nombre_parámetro> <tipo>
```

IN indica que el parámetro es de entrada, OUT de salida e INOUT de entrada/salida. Si el parámetro es de entrada, el procedimiento podrá leer, pero no modificar su valor. Si es de salida, el procedimiento podrá asignarle un valor que será posteriormente leído por el bloque de código que invoca al procedimiento. Si es de entrada/salida, podrá ser leído y modificado.

Las invocaciones a procedimientos se producen mediante el comando **CALL**.

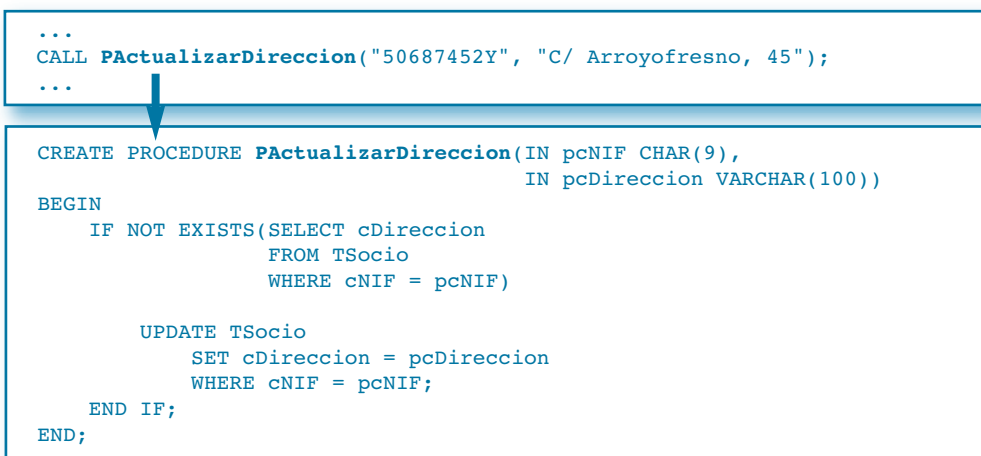


Figura 6.16. Procedimiento almacenado con parámetros de entrada y llamada a dicho procedimiento almacenado. El procedimiento asigna dirección a un socio si no contaba ya con una

```

DECLARE vcPaisEditorial VARCHAR(30);
DECLARE vnPublicaciones INT;
...
CALL PDatosEditorial(32,
                    vcPaisEditorial,
                    vnPublicaciones);
...
    
```

```

CREATE PROCEDURE PDatosEditorial(IN pnEditorialID INT,
                                OUT pcPais VARCHAR(30),
                                OUT pnPublicaciones INT)

BEGIN
    SELECT TPais.cNombre, COUNT(*)
    INTO pcPais, pnPublicaciones
    FROM TEditorial
    INNER JOIN TPais
    ON TEditorial.nPaisID = TPais.nPaisID
    INNER JOIN TLibro
    ON TEditorial.nEditorialID = TLibro.nEditorialID
    WHERE TEditorial.nEditorialID = pnEditorialID
    GROUP BY TPais.cNombre;

END;
    
```

Figura 6.17

Procedimiento almacenado con parámetros de entrada y de salida, y llamada al mismo. Una vez acabada la llamada en la parte invocante, las variables `vcPaisEditorial` y `vnPublicaciones` contendrán los valores que se les asignaron dentro del procedimiento

```

DECLARE vnValor INT;
DECLARE vlCambio BOOLEAN;

...
SET vnValor = 3;

CALL PProcedimiento(vnValor);

SET vlCambio = (vnValor <> 3);
...
    
```

```

...PROCEDURE PProcedimiento
    (IN pnValor INT);
...
SET pnValor = 7;
...
    
```

```

...PROCEDURE PProcedimiento
    (OUT pnValor INT);
...
SET pnValor = 7;
...
    
```

Figura 6.18

Llamada a procedimiento con paso de parámetro. Si se ha declarado el procedimiento como en el código de arriba a la derecha (con el parámetro como IN), el valor de `vlCambio` será FALSE, ya que la asignación de valor dentro del procedimiento no tendrá incidencia sobre el valor del parámetro fuera de dicho procedimiento (`vnValor` seguirá valiendo 3). Si, en cambio, se ha declarado el procedimiento como abajo a la derecha (con el parámetro como OUT), `vnValor` valdrá 7 tras la invocación al procedimiento, por lo que `vlCambio` será TRUE. En este caso el comportamiento sería idéntico declarando el parámetro como OUT o como INOUT, ya que el procedimiento escribe sobre el parámetro, pero no lee su valor

PARA SABER MÁS



La teoría de la programación dice que un procedimiento puede recibir dos tipos de parámetros: por valor y por referencia.

Cuando se recibe un parámetro por valor, lo que se pasa al procedimiento es una copia de la variable original (su valor), por lo que cualquier alteración que sufra el parámetro dentro del procedimiento no quedará patente en la variable original del bloque de código invocador.

En cambio, un parámetro por referencia es una referencia a la variable original, es decir, un puntero a la posición de memoria de dicha variable. Cualquier cambio sobre el valor del parámetro se estará haciendo sobre la variable original.

El paso por valor corresponde con los parámetros IN de SQL/PSM, mientras el paso por referencia se asocia a los parámetros INOUT.



Actividad propuesta 6.9

Basándose en la base de datos de la figura 3.28, crear un procedimiento que reciba como parámetro el código y el nombre de un país y un valor lógico. El procedimiento comprobará si el país ya existe en la tabla TPais, y lo dará de alta si no es así. Al valor lógico, que será leído por el bloque de código invocador a la salida del procedimiento, se le deberá asignar el valor TRUE si el país ya existía y FALSE en caso contrario.

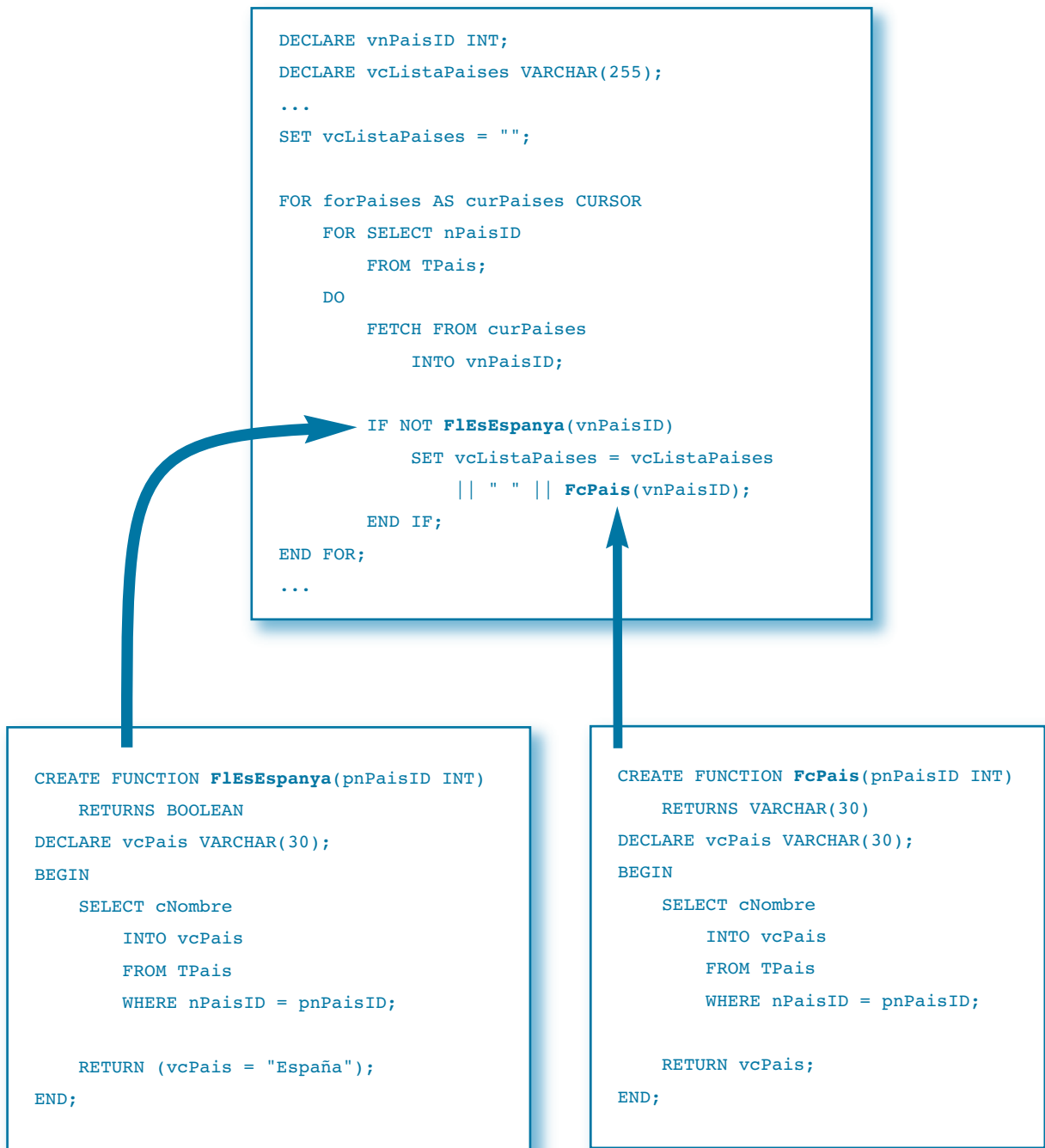
- Las funciones se definen del siguiente modo:

```
CREATE FUNCTION <nombre_función> ([<par1>[, <par2>,...,<parN>]])
    RETURNS <tipo>
    <declaraciones>
BEGIN
    <lista_sentencias>
    RETURN <valor>;
END;
```

Como el propósito de una función es devolver un valor, sus parámetros solo pueden ser de entrada (IN). La declaración de parámetros solo exige indicar su tipo de datos.

La cláusula **RETURNS** indica el tipo de datos del valor que devolverá la función, y la sentencia **RETURN** devolverá dicho valor o expresión. No es obligatorio que **RETURN** sea la última sentencia del bloque. Puede ir en medio del código o incluso aparecer varias veces. A diferencia de otras implementaciones de funciones, las líneas de código que sigan a **RETURN** seguirán ejecutándose hasta el final de la función.

Las funciones son invocadas como expresiones, al igual que ocurría con las funciones integradas descritas en el apartado 5.1.8 del tema anterior.


Figura 6.19

Llamada a dos funciones. El bloque de código superior recorre en un cursor la tabla de países. Por cada uno de ellos invoca a la función de la izquierda para saber si se trata de España. En caso contrario, invoca a la función de la derecha, que le devuelve el nombre del país, y concatena este último con su lista de países. Aunque algo forzado, este ejemplo es muy útil para entender el funcionamiento de las funciones en SQL/PSM



Actividad propuesta 6.10

A partir de la base de datos de la figura 3.28, crear una función que devuelva TRUE si el NIF de socio que recibe como parámetro pertenece a un socio que ha sido dado de alta después del año 2009 y que nunca se ha llevado prestado un libro.



Actividad propuesta 6.11

Reescribir el código de la actividad propuesta 6.9 utilizando una función que devuelva el valor lógico en vez de un procedimiento.

6.2.4. Control de errores. Excepciones

En el cuadro 6.1 se mencionó la variable de sistema SQLSTATE. El tratamiento adecuado de dicha variable es crucial para conocer el estado de las consultas que se ejecutan en un bloque de código, especialmente en lo relativo a la detección de errores y acciones posteriores. Como consecuencia de la ejecución de cualquier línea de código, SQLSTATE puede tomar varios valores:

CUADRO 6.2. Valores de la variable SQLSTATE

Código	Nombre	Descripción
00000		No hay problemas
01000	SQLWARNING	Advertencia
02000	NOT FOUND	No hay registros que devolver
Otros	SQLException	Excepción

Para interceptar un estado concreto habrá que declararlo como condición. Posteriormente podremos referirnos a dicho estado a través del nombre asignado a dicha condición:

```
DECLARE <nombre_estado> CONDITION FOR SQLSTATE <código_estado>;
```

Un uso habitual de SQLSTATE consiste en determinar en qué momento una consulta no devuelve más registros, algo especialmente útil cuando una consulta no devuelve ningún dato o cuando se está recorriendo un cursor en un bucle WHILE o REPEAT. Como el estado 02000 ya está definido en SQL/PSM como NOT FOUND, no es necesario acompañarlo de una sentencia DECLARE. En el siguiente ejemplo se añade el estado 38004 (lectura de datos SQL no permitida) para ilustrar la declaración de una condición:

```

DECLARE vnFacturaID INT;
DECLARE vnImporte, vnImporteAcumulado FLOAT;
DECLARE vcTexto VARCHAR(100);
DECLARE READING_NOT_ALLOWED CONDITION FOR SQLSTATE "38004";

DECLARE curFacturas CURSOR
    FOR SELECT nFacturaID, nImporte
        FROM TFactura
        WHERE dFactura BETWEEN "2013-01-01" AND "2013-12-31"
        ORDER BY dFactura;
BEGIN
    SET vnImporteAcumulado = 0;
    SET vnImporte = 0;

    OPEN curFacturas;

    REPEAT
        SET vnImporteAcumulado = vnImporteAcumulado + vnImporte;

        FETCH FROM curFacturas INTO vnFacturaID, vnImporte;
    UNTIL (vnImporte >= 1000000) OR NOT FOUND OR READING_NOT_ALLOWED
    END REPEAT;

    IF vnImporteAcumulado < 1000000 THEN
        SET vcTexto = "Aún no se ha superado el millón de facturación";
    ELSE
        SET vcTexto = "La factura un millón fue la " || vnFacturaID;
    END IF;

    CLOSE curFacturas;
END;
    
```

Figura 6.20. Código de la figura 6.13, con dos comprobaciones: que el cursor aún tenga registros que devolver en cada iteración y que la lectura esté permitida

Los códigos de excepción definen estados inusuales de los datos que, como se ha visto en el ejemplo, pueden implicar un error de programación (división por cero, consulta de un solo registro que devuelve varios, inserción de valor NULL en un campo NOT NULL...). Si estamos tratando con excepciones, podemos llevar a cabo una tarea extra: establecer un gestor de la excepción (*handler*) para que defina qué tarea ejecutar cuando esta se detecte (EXIT, CONTINUE o UNDO) y qué sentencias llevar a cabo:

```

DECLARE {EXIT|CONTINUE|UNDO} HANDLER
    FOR <código_estado1>[, <código_estado2>,...,<código_estadoN>]
    {<sentencia>|
    BEGIN [ATOMIC]
        <lista_sentencias>;
    END; }
    
```

La cláusula EXIT provoca que el flujo de código se interrumpa y se abandone el bloque de código actual. CONTINUE ignora el error y sigue ejecutando las siguientes sentencias, y UNDO deshace todos los cambios hechos a la base de datos desde el comienzo del bloque de código

(de forma análoga al ROLLBACK de una transacción). Si se define un *handler* con UNDO, el bloque de código debe empezar con una instrucción BEGIN ATOMIC en vez de BEGIN.

```
DECLARE vnFacturaID INT;
DECLARE vnImporte, vnImporteAcumulado FLOAT;
DECLARE vcTexto VARCHAR(100);
DECLARE READING_NOT_ALLOWED CONDITION FOR SQLSTATE "38004";
DECLARE EXIT HANDLER FOR READING_NOT_ALLOWED
BEGIN
    SET vcTexto = "Error de lectura";
    CLOSE curFacturas;
END;

DECLARE curFacturas CURSOR
FOR SELECT nFacturaID, nImporte
FROM TFactura
WHERE dFactura BETWEEN "2013-01-01" AND "2013-12-31"
ORDER BY dFactura;
BEGIN
    SET vnImporteAcumulado = 0;
    SET vnImporte = 0;

    OPEN curFacturas;

    REPEAT
        SET vnImporteAcumulado = vnImporteAcumulado + vnImporte;

        FETCH FROM curFacturas INTO vnFacturaID, vnImporte;
    UNTIL (vnImporte >= 1000000) OR NOT_FOUND OR READING_NOT_ALLOWED
    END REPEAT;

    IF vnImporteAcumulado < 1000000 THEN
        SET vcTexto = "Aún no se ha superado el millón de facturación";
    ELSE
        SET vcTexto = "La factura un millón fue la " || vnFacturaID;
    END IF;

    CLOSE curFacturas;
END;
```

Figura 6.21

Código de la figura 6.20, en el que se incluye un *handler* para la excepción de lectura de datos SQL no permitida. Si salta dicha excepción, se ejecutarán las dos líneas definidas en la declaración del *handler* y se devolverá el flujo de código al bloque de código invocador

Las excepciones también se pueden forzar. En un momento dado puede que al programador le interese que ciertas situaciones se traten como excepciones, aunque técnicamente no lo sean, o bien anticiparse a la generación de la excepción. Para ello se utiliza la sentencia SIGNAL:

```
SIGNAL {<nombre_de_condición>|SQLSTATE <código_estado>};
```

```

DECLARE vnFacturaID INT;
DECLARE vnImporte FLOAT;
DECLARE DIVISION_BY_ZERO CONDITION FOR SQLSTATE "22012";
DECLARE EXIT HANDLER FOR DIVISION_BY_ZERO
    CLOSE curFacturas;

BEGIN
    FOR forFacturas AS curFacturas CURSOR
        FOR SELECT nFacturaID, nImporte
            FROM TFactura
            WHERE dFactura BETWEEN "2013-01-01" AND "2013-12-31"
        DO
            FETCH FROM curFacturas
                INTO vnFacturaID, vnImporte;

            IF vnImporte = 0 THEN
                SIGNAL DIVISION_BY_ZERO;
            END IF;

            ...

        END FOR;
    END;

```

Figura 6.22. Se recorren las facturas del año 2013 para su tratamiento posterior. Si alguna factura tiene importe cero, se fuerza la excepción 22012 (división por cero), aunque aún no esté ocurriendo

Actividad propuesta 6.12

Modificar el código de la figura 6.21 añadiendo un *handler* para la condición “Valor de parámetro no válido” (código 22023). El *handler* asignará el valor cero a la variable *vnImporteAcumulado* y el texto “Error de parámetro” a *vcTexto*, cerrará el cursor y devolverá el control al bloque de código invocante.



6.2.5. Triggers

En el capítulo 3 se abordó el concepto de integridad referencial y se vieron varios mecanismos para gestionarla. Un paso más allá consiste en la posibilidad de crear *triggers*, es decir, bloques de código que se ejecutan en el momento exacto en que cambia el contenido de la base de datos. Los *triggers* (traducidos a veces como “disparadores”) se definen por cada tabla, y se puede declarar uno distinto por cada uno de los siguientes instantes en el tiempo:

- Justo antes de producirse un **INSERT**.
- Justo después de producirse un **INSERT**.
- Justo antes de producirse un **UPDATE**.
- Justo después de producirse un **UPDATE**.
- Justo antes de producirse un **DELETE**.
- Justo después de producirse un **DELETE**.

La sintaxis de declaración de un *trigger* es la siguiente:

```
CREATE TRIGGER <nombre_trigger>
  {BEFORE|AFTER}
  {INSERT|DELETE|UPDATE [OF <campo1>[, <campo2>,..., <campoN>]]}
  ON <nombre_tabla>
  [REFERENCING OLD {TABLE|ROW} AS <alias_tabla_o_registro_antiguo>]
  [REFERENCING NEW {TABLE|ROW} AS <alias_tabla_o_registro_nuevo>]
  [FOR EACH {ROW|STATEMENT}]
  {<sentencia>|
  BEGIN ATOMIC
    <lista_sentencias>;
  END;}
```

Si se indica **BEFORE**, el código del *trigger* se ejecutará justo antes de que se lleve a cabo la inserción, actualización o borrado; si se indica **AFTER**, se ejecutará después.

Las acciones posibles son **INSERT**, **UPDATE** o **DELETE**. En el caso de **UPDATE**, se puede especificar el nombre de uno o varios campos, de modo que el *trigger* solo se ejecute si se ha modificado alguno de esos campos, pero no los demás. Si no se indican nombres de campo, el *trigger* se ejecutará ante cualquier sentencia **UPDATE** sobre la tabla.

Un *trigger* se puede definir por registro (**FOR EACH ROW**) o por sentencia de actualización (**FOR EACH STATEMENT**). Si se define por sentencia (la opción por defecto), se ejecutará una sola vez por cada **INSERT**, **UPDATE** o **DELETE**. Si se define por registro, se ejecutará una vez por cada registro afectado (en un **UPDATE** masivo, por ejemplo).

En un *trigger* por sentencia se pueden asignar dos alias a la tabla en cuestión: uno para la tabla tal cual estaba antes de la inserción, actualización o borrado (**OLD TABLE AS <alias_tabla_antigua>**) y otro para la tabla tal cual quedará después de la ejecución de la sentencia (**NEW TABLE AS <alias_tabla_nueva>**). Si el *trigger* es por registro, se pueden asignar alias al registro en sus dos estados: antes de la ejecución de la sentencia (**OLD ROW AS <alias_registro_antiguo>**) y después (**NEW ROW AS <alias_registro_nuevo>**). De esta forma, dentro del código del *trigger* se puede acceder, por ejemplo, a los valores que tenían los campos de un registro antes de ejecutar un **UPDATE** sobre él y después de ejecutar dicho **UPDATE**.

En los próximos ejemplos, y siguiendo la adaptación a la notación húngara presente a lo largo de este libro, prefijaremos el nombre de cada *trigger* con las letras “trg” (minúscula).

```
CREATE TRIGGER trgAutorInsBfr
    BEFORE INSERT ON TAuthor
    REFERENCING NEW ROW AS rowNuevo
    FOR EACH ROW
    BEGIN ATOMIC
        IF EXISTS(SELECT *
            FROM TAuthor
            WHERE TAuthor.cNombre = rowNuevo.cNombre
            AND TAuthor.cApellidos = rowNuevo.cApellidos) THEN
            SIGNAL SQLSTATE "22028";
        END IF;
    END;
```

Figura 6.23. *Trigger* antes de realizar un INSERT en TAuthor. Si el nombre y los apellidos del autor ya existen, se fuerza una excepción 22028 (el registro ya existe). El alias rowNuevo sirve para poder acceder a los valores que se van a insertar en los campos cNombre y cApellidos

```
CREATE TRIGGER trgSocioUpdAft
    AFTER UPDATE ON TSocio
    REFERENCING OLD ROW AS rowAntiguo,
    NEW ROW AS rowNuevo
    FOR EACH ROW
    BEGIN ATOMIC
        IF rowNuevo.cDireccion <> rowAntiguo.cDireccion THEN
            UPDATE TSocio
            SET TSocio.cDireccion = rowNuevo.cDireccion || " [NEW]"
            WHERE TSocio.cNIF = rowAntiguo.cNIF;
        END IF;
    END;
```

Figura 6.24. *Trigger* tras realizar un UPDATE en TSocio. Si la dirección es nueva (distinta a la anterior) se le añade el texto "[NEW]"

Los *triggers* BEFORE y AFTER tanto del tipo FOR EACH ROW como FOR EACH STATEMENT pueden convivir, de modo que un simple UPDATE puede desencadenar dos *triggers* (los FOR EACH STATEMENT) y otros dos por cada registro afectado (los FOR EACH ROW). En caso de que ocurra esta situación, el orden de ejecución de *triggers* es:

1. BEFORE + FOR EACH STATEMENT.
2. Para cada registro afectado:
 1. BEFORE + FOR EACH ROW.
 2. Sentencia INSERT, UPDATE o DELETE.
 3. AFTER + FOR EACH ROW.
3. AFTER + FOR EACH STATEMENT.

No obstante, la acumulación de *triggers* ralentiza el funcionamiento de la base de datos y es fuente de numerosos problemas derivados de efectos colaterales, por lo que no es recomendable abusar de ellos.



Actividad propuesta 6.13

Una aplicación habitual de los *triggers* son las funciones de auditoría. En muchas bases de datos de empresas e instituciones que albergan información sensible es muy habitual añadir a todas las tablas campos en los que almacenar la fecha y hora de la última actualización (gracias al tipo de datos `TIMESTAMP` y a la variable `CURRENT_TIMESTAMP` comentada anteriormente) y el usuario que la realizó (gracias a la variable `CURRENT_USER`), entre otros datos. A veces se crea una estructura paralela donde cada tabla cuenta con otra tabla hermana dedicada exclusivamente a labores de auditoría, de modo que se guardará la traza de cualquier cambio sobre los datos, por pequeño que sea. Del mismo modo, los borrados pueden ser lógicos en vez de físicos. En ese caso, se añade un campo lógico a cada tabla que indique si el registro está borrado o no; obviamente, para filtrar los registros marcados como borrados, en todas las `SELECT` habrá que incluir la cláusula “`WHERE NOT IBorrado`”.

Esta actividad tiene dos partes:

1. Ampliar la tabla `TSocio` de la base de datos de la figura 3.28 con los siguientes campos:
 - `IBorrado` (`BOOLEAN`). Indica si el registro está borrado o no.
 - `dNuevo` (`TIMESTAMP`). Fecha/hora de alta del registro.
 - `cUsrNuevo` (`VARCHAR(255)`). Usuario que dio de alta el registro.
 - `dUltimaModificacion` (`TIMESTAMP`). Fecha/hora de última modificación del registro.
 - `cUsrUltimaModificacion` (`VARCHAR(255)`). Usuario que modificó el registro por última vez.
 - `dBorrado` (`TIMESTAMP`). Fecha/hora de borrado del registro.
 - `cUsrBorrado` (`VARCHAR(255)`). Usuario que borró el registro.
2. Programar los *triggers* necesarios para llevar a cabo la auditoría de esta tabla. Habrá que declarar un *trigger* por cada inserción de registro, otro por cada actualización de registro y otro por cada borrado de registro.

Resumen

La existencia de un lenguaje de programación integrado en los SGBD permite disminuir el tráfico de red entre la base de datos y el *software* que conecta con ella. También permite agrupar las funcionalidades relacionadas con los datos e independizarlas del resto de procesos. Poder ejecutar bloques de código en el servidor facilita las tareas de administración, ya que permite programar y temporizar *backups*, generación de estadísticas, gestión de usuarios, etc.

Aunque cada SGBD define su propio lenguaje de programación, desde 1999 existe un estándar llamado SQL/PSM. Su gestión de variables, funciones, procedimientos y estructuras de control es similar a la de otros lenguajes de programación. Cabe destacar el uso de cursores (estructuras de almacenamiento en memoria que contienen el resultado de una consulta).

Un elemento muy importante en todo SGBD son los *triggers*, bloques de código que se ejecutan a partir de una acción sobre los datos (inserción, actualización o borrado).

EJERCICIOS PROPUESTOS

1. Buscar información sobre los lenguajes procedimentales de base de datos más utilizados y comparar su sintaxis con la de SQL/PSM.
2. Utilizando la variable de sistema `CURRENT_DATE`, crear una función SQL/PSM que compruebe si una persona es mayor de edad (18 años). La función tendrá la siguiente declaración:

```
CREATE FUNCTION FlMayorDeEdad(pdNacimiento DATE)
    RETURNS BOOLEAN
```

3. Sobre la base de datos de la figura 3.28, crear una función que reciba un NIF de socio y devuelva una cadena de caracteres con sus datos respetando el formato siguiente:

```
09653801 - Lázaro Cardenal, Ruth (C/ Arapiles, 14). Tfno: 658552489.
```

4. Modificar el código de la figura 6.20 para que incluya un *handler* correspondiente a las advertencias (SQLWARNING, código 01000). El *handler* deberá ignorar los *warnings*, pero debe añadir el texto " [WRN]" a la variable `vcTexto`.
5. Escribir un *trigger* que lleve a cabo la siguiente tarea sobre los borrados de la tabla `TLibro`. Cada vez que se borre un libro se comprobará si era el único libro de su editorial. En caso afirmativo, se borrará la editorial.

ACTIVIDADES DE AUTOEVALUACIÓN

1. En SQL/PSM hay que abrir y cerrar los cursores:
 - ☐ a) En todas las situaciones.
 - ☐ b) Siempre, excepto cuando se tratan dentro de un bucle.
 - ☐ c) Siempre, excepto cuando se tratan dentro de un bucle FOR.
2. Los bucles WHILE:
 - ☐ a) Evalúan la condición de salida al principio.
 - ☐ b) Evalúan la condición de salida al final.
 - ☐ c) Evalúan la condición de permanencia al principio.
3. Los parámetros de una función:
 - ☐ a) Pueden ser de entrada, salida o entrada/salida.
 - ☐ b) Pueden ser de entrada o salida.
 - ☐ c) Siempre son de entrada.
4. Para saber si un cursor no tiene más registros que mostrar:
 - ☐ a) Hay que preguntar si el código de estado es el 02000.
 - ☐ b) Hay que ejecutar una función sobre el cursor.
 - ☐ c) Hay que evaluar las variables rellenas por la sentencia FETCH.
5. Un *trigger* de tipo BEFORE sobre un UPDATE:
 - ☐ a) Se ejecuta tras la sentencia UPDATE correspondiente.
 - ☐ b) Se ejecuta antes de la sentencia UPDATE correspondiente.
 - ☐ c) No se puede crear un *trigger* de tipo BEFORE sobre un UPDATE.

SOLUCIONES:

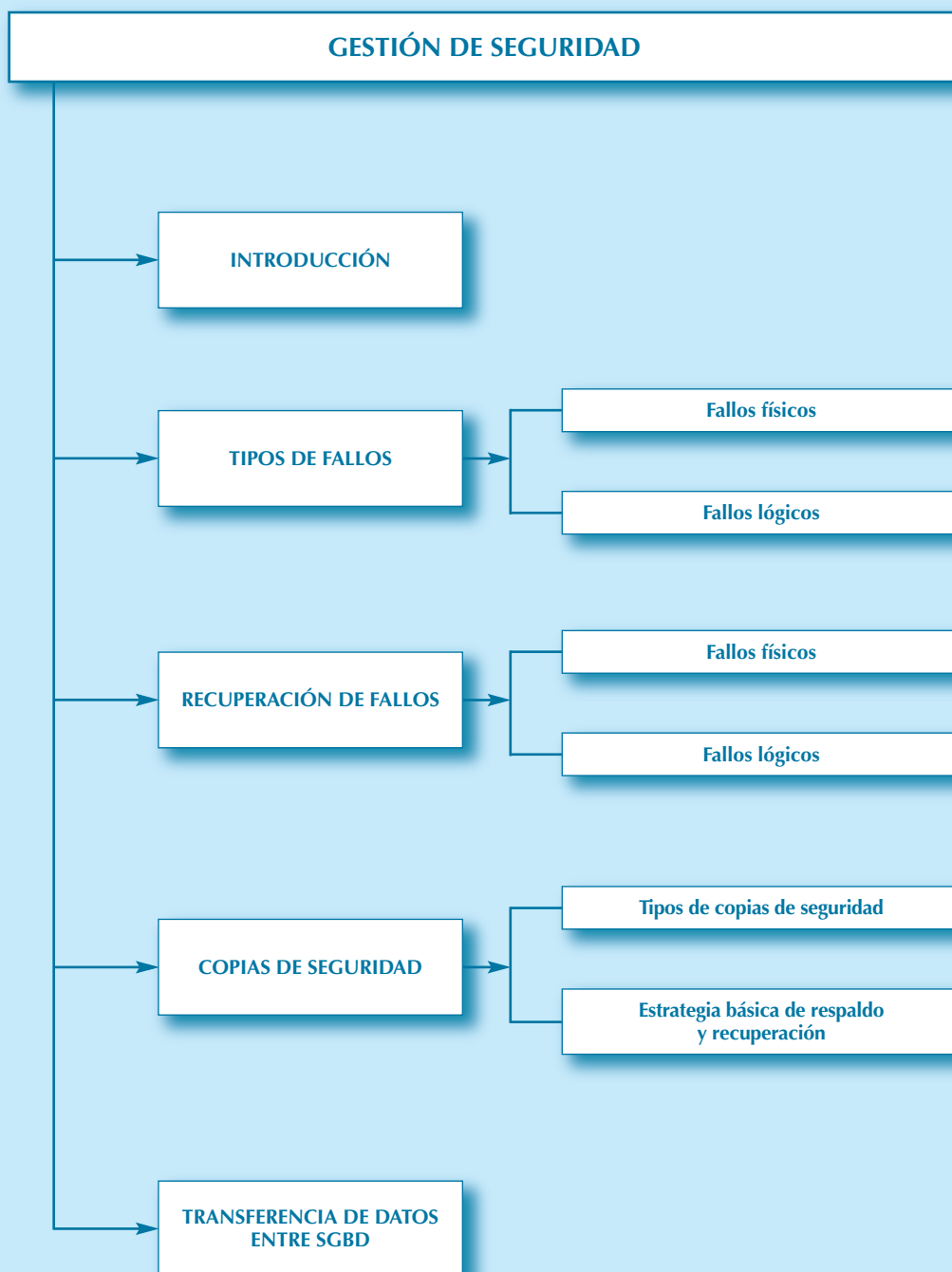
1. ☐ a ☐ b ☒ c2. ☐ a ☐ b ☒ c3. ☐ a ☐ b ☒ c4. ☒ a ☐ b ☐ c5. ☐ a ☒ b ☐ c

Gestión de seguridad

Objetivos

1. Entender por qué es necesario contar con una gestión de seguridad.
2. Discriminar los fallos físicos de los lógicos.
3. Conocer cómo luchar contra dichos tipos de fallos.
4. Asimilar la necesidad de efectuar copias de seguridad.
5. Utilizar correctamente los distintos tipos de *backup* existentes.
6. Familiarizarse con la problemática de la transferencia de datos entre sistemas gestores.

Mapa conceptual del capítulo



Glosario

Cifrar. Codificar un dato para que resulte ilegible. Para recuperar el dato original habrá que ejecutar un proceso de descifrado.

Entorno de producción. Equipos, líneas de comunicaciones, *software* y bases de datos donde se explotan sistemas de información. Se incide en el término “producción” para establecer diferencias con el “entorno de desarrollo” y el “entorno de pruebas”, implantaciones del sistema de información destinadas a su construcción y pruebas, respectivamente, y donde la incidencia de posibles errores es menor.

Fichero de log (traducido a veces como “cuaderno de bitácora”). Registro de eventos acaecidos en un sistema informático.

Migración. Proceso de transferir información entre distintos tipos o formatos de almacenamiento físico o lógico.

Parche (patch). Paquete de actualización de un *software*.

N/A (not applicable). No aplicable. Que no tiene vigencia en el contexto actual.

7.1. Introducción

Las bases de datos almacenan el elemento más valioso de toda organización empresarial: la información. Gran parte del éxito en la consecución de los objetivos de una empresa radica en una adecuada gestión de la seguridad de dichos datos.

Como norma básica se deben cumplir las cinco características de seguridad agrupadas bajo las siglas CIDAN:

- *Confidencialidad.* Ningún usuario no autorizado debe ser capaz de acceder a la información.
- *Integridad.* La información ha de permanecer inalterable.
- *Disponibilidad.* La información debe estar accesible en un espacio de tiempo razonable.
- *Autenticación.* Solo los usuarios autorizados, previa demostración de su identidad, serán capaces de acceder a la información.
- *No repudio.* Ningún usuario podrá negar su identidad ni, por tanto, su autoría en la consulta o la modificación de la información.

En este tema nos vamos a centrar en la disponibilidad de la información y en cómo garantizarla mediante diversos mecanismos.



Figura 7.1

La gestión de la seguridad es un elemento central en las bases de datos

PARA SABER MÁS



Una forma de garantizar la característica de no repudio (también llamada *irrenunciabilidad*) es establecer certificados digitales, de modo que toda información insertada, modificada o borrada de la base de datos lleve la firma digital del usuario que lleva a cabo la acción.

7.2. Tipos de fallos

En el ámbito de la seguridad hay varias clasificaciones de fallos. Una de ellas diferencia los internos (causados dentro de la organización) de los externos (ataques, causas meteorológicas, etc.). No obstante la clasificación más útil distingue típicamente entre los fallos físicos y lógicos.



Actividad propuesta 7.1

Antes de leer los dos apartados siguientes, elaborar una lista de fallos potenciales que pueden afectar a la información almacenada en una base de datos.

7.2.1. Fallos físicos

Son los derivados de problemas en el *hardware* o en las instalaciones en las que se encuentran los equipos:

- Averías en discos duros.
- Averías en servidores.
- Mal funcionamiento de la red.
- Problemas en suministros (cortes de electricidad).
- Incendios, inundaciones, accidentes.
- Ataques físicos.
- Robos.

En algunas ocasiones estos errores tienen que ver con el acceso físico indebido a la sala donde se encuentran los servidores de bases de datos, por lo que la competencia sobre este tipo de problemas se comparte con los responsables de seguridad del edificio y con el departamento de seguridad, si existe.



Figura 7.2
Centro de proceso
de datos (CPD)

7.2.2. Fallos lógicos

Tienen que ver con incidencias en el *software*:

- Errores de diseño de la base de datos.
- Falta de actualización de versiones del SGBD o del sistema operativo.
- Errores en las aplicaciones que acceden a la base de datos.
- Errores humanos de manipulación de la base de datos.
- Ataques lógicos.
- Virus.
- Mala gestión de seguridad lógica.

Un problema habitual tiene que ver con las contraseñas de aplicaciones almacenadas en tablas de bases de datos. Dichas contraseñas deben ir siempre cifradas, de modo que ningún informático no autorizado pueda recuperarlas mediante una simple consulta **SELECT**. Varios SGBD ofrecen un tipo de datos especial para almacenar contraseñas cifradas. De no ser así, el personal de desarrollo tendrá que codificarlas antes de almacenarlas en la base de datos.

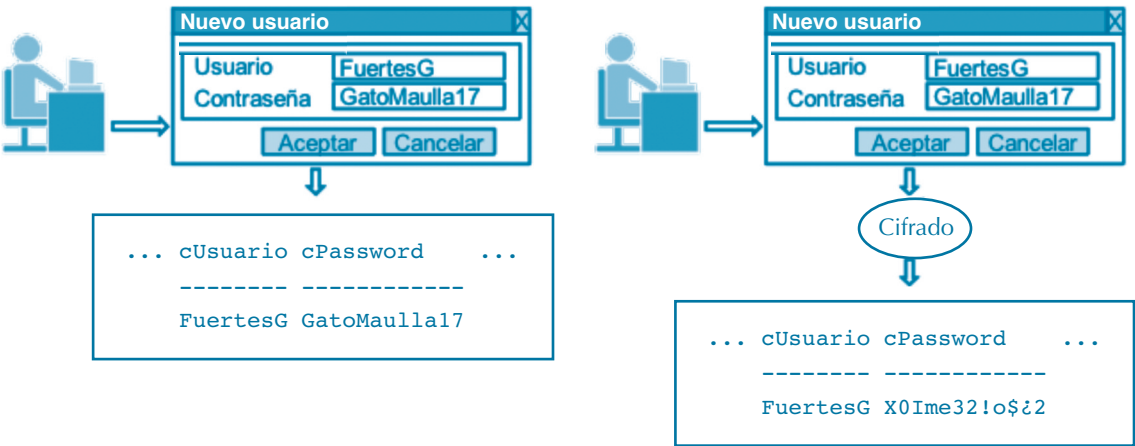


Figura 7.3
Alta de una contraseña en una base de datos sin cifrar (izquierda) y cifrada (derecha)

7.3. Recuperación de fallos

La máxima más importante en el ámbito de la seguridad informática es que dicha seguridad no se puede garantizar. Por mucho que se extremen las precauciones, siempre habrá causas potenciales de inestabilidad. No obstante, toda organización debe realizar una gestión de la seguridad tan concienzuda como sea posible.



Actividad propuesta 7.2

Antes de leer los dos apartados siguientes, elaborar una lista con las potenciales soluciones para los fallos vistos en los apartados 7.2.1 y 7.2.2.

7.3.1. Fallos físicos

En muchos casos los fallos físicos derivan de la falta de prevención. De cara a disminuir el volumen de tales problemas es recomendable tener en cuenta los siguientes elementos en la fase de diseño de nuestros sistemas de información:

- *Sistemas redundantes de discos.* Todo servidor de base de datos que se precie debe contar con una matriz redundante de discos independientes (RAID, *redundant array of independent disks*). Esta técnica permite que varios discos duros trabajen como un solo disco lógico, almacenando los datos de forma redundante. Ante un error en un disco, los demás seguirán funcionando y no se perderá información.



Figura 7.4. Sistema de discos redundante (RAID)

- *Elección acertada de servidores y contratos de mantenimiento.* Los servidores suelen ser los equipos más importantes en una empresa. Las limitaciones en inversión económica y la falta de precisión en la determinación de las necesidades pueden ocasionar serios problemas a largo plazo. Es importante elegir el número de servidores necesarios y sus características. Del mismo modo, la cobertura de los contratos de mantenimiento firmados con el proveedor del *hardware* deben definir responsabilidades y tomas de acción en caso de problemas.
- *Gestión de información redundante.* Es recomendable la existencia de un centro de contingencia de datos (CDC, *contingency data center*) situado a varios kilómetros del centro de proceso de datos (CPD) en el que se encuentran los servidores de bases de datos. En dicho CDC se replicarán el *hardware* y el *software* más importantes de la oficina principal. Tanto los datos como las versiones de *software* deben estar siempre actualizados en el CDC para poder seguir trabajando desde allí en caso de que el CPD quede inoperativo.
- *Gestión de backups.* Se deben programar con frecuencia copias de seguridad de las bases de datos de la organización. Dichos *backups* deben estar fácilmente disponibles para su restauración en los servidores de bases de datos en caso de necesidad, pero también deben extraerse a soportes de almacenamiento secundario (cinta magnética, DVD) guardados en armarios ignífugos y trasladados al CDC.
- *Mirroring.* La técnica de *mirroring* consiste en replicar la información en tiempo real a un servidor redundante, es decir, escribir los datos en dos servidores a la vez. Si el servidor principal presentase errores físicos, el secundario (típicamente situado en el CDC) dispondría de la información totalmente actualizada. Obviamente, implantar *mirroring* implica mayores tiempos de retardo en la escritura de la información, así como una gestión más cara y compleja de las líneas de comunicaciones.



Figura 7.5. Proceso de *mirroring* al actualizar un usuario la información

- *Plan de contingencia.* Toda organización debe definir un procedimiento a seguir en caso de problemas de fuerza mayor (cortes en suministros eléctricos o comunicaciones con el exterior, fenómenos climatológicos, etc.). Dicho plan debe contemplar qué acciones tomar con respecto al acceso a las bases de datos de la compañía, ya sea en el CPD o en el CDC.
- *Ubicación física.* El CPD debe estar situado en una zona del edificio ajena a los riesgos climáticos y geológicos (terremotos, inundaciones), con restricción de acceso y no publicitado. También ha de contar con mecanismos de detección y extinción de incendios. Los elementos de seguridad física habrán de coordinarse con el departamento de seguridad de la empresa, en caso de existir, así como con los responsables de seguridad del edificio.



Figura 7.6
Acceso a un centro de proceso de datos (CPD)

- *Acuerdo en la contratación de suministros.* Toda organización debe minimizar la incidencia de condicionantes externos. De nada sirve llevar a cabo una gestión modélica de servicios y procesos si la compañía telefónica, el proveedor de acceso a Internet o la empresa de electricidad cortan el suministro por problemas ajenos a nuestra empresa. Es básico contar

con las compañías más fiables y suscribir contratos que incluyan cláusulas de calidad de servicio (QoS, *quality of service*) en las que se establezcan con detalle la responsabilidad del proveedor y las consecuencias legales derivadas de infracciones en la prestación del servicio.

- *Pólizas de seguros*. Como todo elemento valioso, la información también está sujeta a la contratación de pólizas de seguros que puedan indemnizar a la organización en casos concretos (robos, incendios).

Actividad propuesta 7.3

Consultar en las páginas web de las principales compañías de teléfono e Internet las condiciones de sus ofertas a medida para grandes empresas.



7.3.2. Fallos lógicos

Parte de la gestión de fallos lógicos tiene que ver con un buen diseño y mantenimiento de los recursos *software* de la organización, pero el problema se complica cuando los fallos tienen su origen en ataques, ya que la pericia de *hackers* y *crackers* puede imponerse a los mecanismos de seguridad. Es recomendable hacer hincapié en los siguientes aspectos:

- *Acceso al servicio*. Se debe impedir el acceso al servicio de base de datos a los equipos de la oficina y/o del exterior que no necesiten acceder a la información. Esta gestión suele corresponder al departamento de redes e infraestructuras.
- *Acceso al servidor*. Tanto el servidor como el SGBD deben estar protegidos por usuario y contraseña.
- *Autenticación en el SGBD*. No se debe permitir el acceso al SGBD a ningún usuario que no esté dado de alta en el catálogo de usuarios.
- *Autenticación en el sistema operativo*. En muchas ocasiones, en vez de crear usuarios en el SGBD se utiliza una gestión de permisos basada en los usuarios y perfiles del sistema operativo.
- *Gestión de perfiles y usuarios*. Se debe afinar mucho en la definición de permisos por usuario. Algunos tendrán acceso a todas las bases de datos y tablas, otros solo a alguna. Unos usuarios podrán modificar información y otros solo verla. Aunque haya quien tenga acceso total a la información contenida en las tablas, solo determinados usuarios podrán administrar el SGBD (mantenimiento de bases de datos y usuarios, gestión de *backups*, temporización de eventos).

Un criterio habitual a la hora de establecer permisos a usuarios y perfiles de usuario se resume en la frase “todo lo que no está permitido debe estar expresamente prohibido”. Toda organización eliminará problemas potenciales si cumple esta máxima, denegando por defecto todos los accesos a la base de datos y concediendo solamente los necesarios.

PARA SABER MÁS



Aunque no hay unanimidad dentro de la comunidad informática, el término *hacker* suele aplicarse a las personas que consiguen cruzar barreras de seguridad informática por diversión o satisfacción personal, mientras la palabra *cracker* se reserva a quien consigue traspasar dichos límites con fines lucrativos.

7.4. Copias de seguridad

Una copia de seguridad (también llamada copia de respaldo o, más comúnmente, *backup*) es un duplicado de la información de una base de datos. El objetivo de contar con copias de seguridad es poder restaurarlas al entorno de producción si este sufriese algún problema.

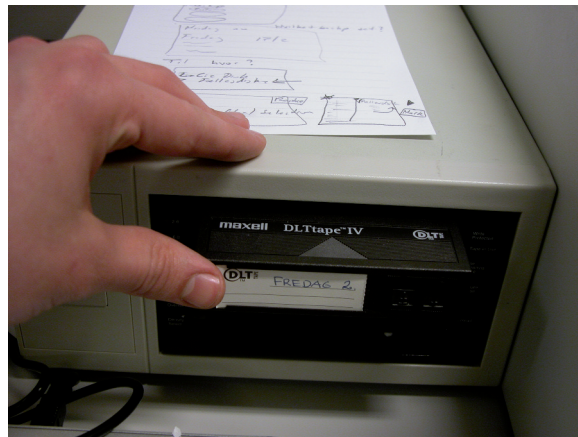


Figura 7.7
Sistema de *backup*

7.4.1. Tipos de copias de seguridad

Se definen varios tipos de *backups* en función de cómo se lleva a cabo la copia de la información de una base de datos para almacenarla como respaldo:

- *En frío* (consistentes). Se realizan con el servicio de base de datos detenido, por lo que no puede haber transacciones en curso. El estado de la información es, por tanto, consistente, y se puede crear un *backup* que corresponda a una copia física de todos los ficheros que conforman la base de datos.
- *En caliente* (inconsistentes). Se realizan con el servicio activo, lo que afecta a su rendimiento, con la posibilidad de que se ejecuten transacciones durante el *backup*. Se efectúan cuando no hay posibilidad de detener el servicio de base de datos.

También se incluyen en esta categoría los *backups* de bases de datos que han sufrido un cierre anormal (aunque el servicio esté detenido, los datos presentan un estado inconsistente).

- *Incrementales.* Solamente se almacenan los datos que han variado desde el último *backup* completo. Es más rápido y afecta menos al rendimiento del servidor que el *backup* completo en caliente, pero ralentiza el proceso de restauración, ya que primero hay que restaurar el último *backup* completo que se hizo y después ir restaurando uno a uno todos los *backups* incrementales que se llevaron a cabo desde entonces.
- *Lógicas.* No son *backups* como tales. Se basan en repetir las operaciones que llevaron a la base de datos a un estado concreto, bien mediante importación de datos o mediante ejecución de ficheros de *log* donde queda almacenada cada inserción, actualización o borrado en tablas de la base de datos. Generalmente se centran en la información de la base de datos, y no incluyen información de control.

Actividad propuesta 7.4

Comprobar en Internet qué herramientas de *backup* ofrecen los SGBD más populares. Crear y restaurar *backups* con alguno de ellos.



7.4.2. Estrategia básica de respaldo y recuperación

Generalmente el servicio de base de datos solamente se detiene en circunstancias concretas (actualización de la versión del SGBD, instalación de parches del sistema operativo, modificación de *hardware* del servidor). En esos momentos se puede hacer un *backup* en frío. En la mayoría de los casos el *backup* será en caliente.

Es recomendable ejecutar un *backup* completo de cada base de datos todos los días. Generalmente el momento elegido para llevar a cabo dicho *backup* es durante la madrugada, debido a la ausencia de usuarios. También se deben hacer *backups* incrementales durante el día, intentando temporizarlos en momentos de poca actividad, especialmente al final de procesos de actualización.

Los *backups* se pueden llevar a cabo sobre el mismo disco duro del servidor, en una máquina externa (en ese caso se ralentizará el tráfico en la red) o en otro equipo en una SAN (*storage area network*, red de área de almacenamiento). Una vez volcados los datos, se deben guardar en soportes de almacenamiento externos.



Figura 7.8. Los *backups* se pueden realizar en máquinas externas

7.5. Transferencia de datos entre SGBD

Cambiar de SGBD no es tarea fácil. Aunque todos parten del mismo estándar, cada compañía ha desarrollado su propia arquitectura, su propio sistema de almacenamiento en memoria secundaria y sus propios tipos de datos.

CUADRO 7.1. Tipos de datos de los SGBD(O)R más populares

	Oracle	MySQL	Microsoft SQL Server	PostgreSQL
Cadenas de caracteres	CHAR, VARCHAR2, CLOB, NCLOB, NVARCHAR2, NCHAR, LONG (obsoleto)	CHAR, BINARY, VARCHAR, VARBINARY, TEXT, TINYTEXT, MEDIUMTEXT, LONGTEXT	CHAR, VARCHAR, TEXT, NCHAR, NVARCHAR, NTEXT	CHAR, VARCHAR, TEXT
Números exactos	NUMBER	TINYINT (8 bits), SMALLINT (16 bits), MEDIUMINT (24 bits), INT (32 bits), BIGINT (64 bits), DECIMAL	TINYINT, SMALLINT, INT, BIGINT, NUMERIC, DECIMAL, SMALLMONEY, MONEY	SMALLINT (16 bits), INTEGER (32 bits), BIGINT (64 bits), DECIMAL, NUMERIC
Números aproximados	BINARY_FLOAT, BINARY_DOUBLE	FLOAT (32 bits), DOUBLE (también llamado REAL) (64 bits)	FLOAT, REAL	REAL (32 bits), DOUBLE PRECISION (64 bits)
Fechas, horas e intervalos	DATE, TIMESTAMP (con/sin TIMEZONE), INTERVAL	DATETIME, DATE, TIMESTAMP, YEAR	DATE, DATETIMEOFFSET, DATETIME2, SMALLDATETIME, DATETIME, TIME, TIMESTAMP	DATE, TIME (con/sin TIMEZONE), TIMESTAMP (con/sin TIMEZONE), INTERVAL
Valores lógicos	N/A	BIT(1), BOOLEAN (también llamado BOOL) = sinónimo de TINYINT	BIT	BOOLEAN
Objetos binarios	BLOB, RAW, LONG RAW (obsoleto), BFILE	TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB	BINARY, VARBINARY, IMAGE, FILESTREAM	BYTEA
Otros	SPATIAL, IMAGE, AUDIO, VIDEO, DICOM, XMLType	ENUM, SET, tipos de datos GIS	CURSOR, HIERARCHYID, UNIQUEIDENTIFIER, SQL_VARIANT, XML, TABLE	ENUM, POINT, LINE, LSEG, BOX, PATH, POLYGON, CIRCLE, CIDR, INET, MACADDR, BIT, UUID, XML, JSON, arrays, composites, rangos, tipos definidos por el usuario

Según lo especificado en el cuadro anterior, los cuatro gestores cuentan con tipos DATE, pero Oracle y MySQL carecen de tipo TIME, como exige el estándar. SQL Server y PostgreSQL incluyen un tipo TEXT que no existe en los otros dos gestores. En Oracle no existe el tipo BOOLEAN y en SQL Server se llama BIT. Aunque los gestores suelen ofrecer herramientas de importación, exportación y transferencia de datos desde otros SGBD, obviamente tomar una base de datos creada en un sistema y transferirla a otro es un proceso arduo y complejo. En la mayoría de los casos requiere de un proceso de *migración*, que implica crear un *software* específico para llevar a cabo la transferencia siguiendo los pasos dictaminados por la ingeniería del *software* (educación de requisitos, elección de ciclo de vida, análisis, diseño, programación, pruebas y ejecución, entre otros).

Actividad propuesta 7.5

Explorar las herramientas de transferencia de datos entre sistemas que ofrecen los SGBD más populares. Intentar transferir pequeñas bases de datos entre gestores.



Resumen

Como objeto más valioso de toda organización, la información está sujeta a diversos tipos de amenazas, por lo que es necesario contar con una gestión de seguridad que afecte a las bases de datos y su entorno (el SGBD, el sistema operativo, los servidores, accesos de red, etc).

La clasificación de fallos más habitual es la que diferencia entre fallos físicos (efectuados sobre el *hardware* o las líneas de comunicaciones) y lógicos (ataques cibernéticos, virus, etc.). Para luchar contra este tipo de amenazas es obligatorio contar con una infraestructura bien diseñada, así como con una buena política de duplicación de datos (*mirroring*, *backups*) y acuerdos legales con terceros.

En lo referente a las copias de seguridad, hay que diseñar un procedimiento preciso que permita efectuarlas sin afectar al servicio y restaurarlas de forma eficiente cuando sea necesario. Se deben conocer perfectamente los distintos tipos de *backup* existentes y las herramientas que ofrece cada SGBD para su gestión.

Otro aspecto importante es el referente a la transferencia de información entre SGBD, una tarea especialmente compleja debido a las diferencias de diseño entre los sistemas de las distintas compañías.

ACTIVIDADES DE AUTOEVALUACIÓN

1. Una de las principales normas de seguridad CIDAN es:
 - ☐ a) La autenticación.
 - ☐ b) La atomicidad.
 - ☐ c) La integración.
2. Entre los fallos físicos se encuentran:
 - ☐ a) Los virus.
 - ☐ b) Los robos.
 - ☐ c) Los errores de diseño.
3. A la sala de ordenadores principal de una empresa se la conoce como:
 - ☐ a) CDC (centro de contingencia de datos).
 - ☐ b) CD (centro de datos).
 - ☐ c) CPD (centro de proceso de datos).
4. Un buen sistema de seguridad debe prohibir el acceso:
 - ☐ a) A todos los usuarios, excepto a los que necesiten acceder al servicio.
 - ☐ b) A todos los usuarios externos a la organización.
 - ☐ c) A usuarios concretos tras un proceso de identificación de los mismos.
5. Para restaurar un *backup* incremental:
 - ☐ a) Basta con ejecutar el proceso de restauración sobre dicho *backup*.
 - ☐ b) Hay que restaurar ese *backup* y los *backups* incrementales anteriores.
 - ☐ c) Hay que restaurar el último *backup* completo y todos los *backups* incrementales posteriores.

SOLUCIONES:

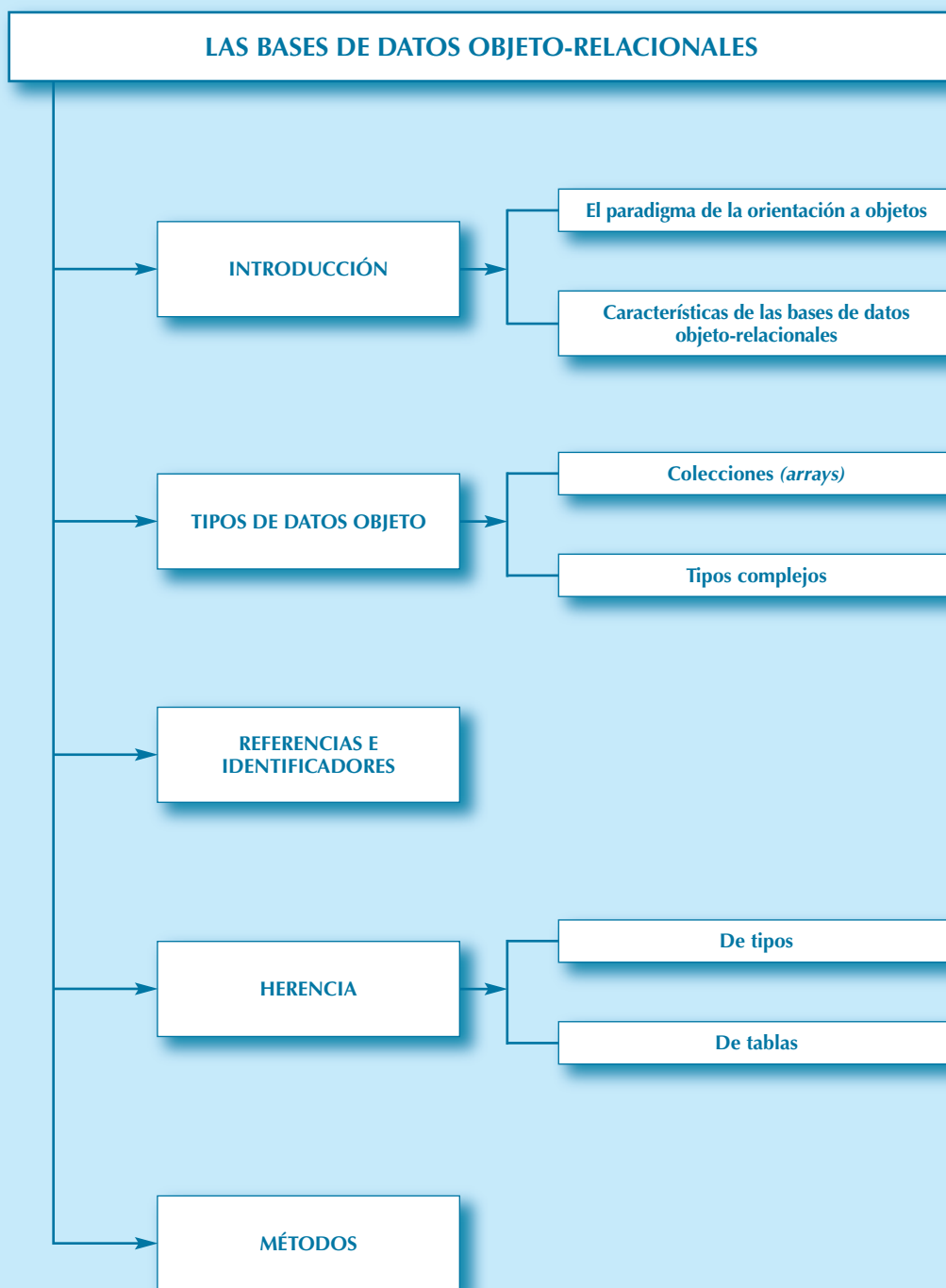
1. ☒ a ☐ b ☐ c2. ☐ a ☒ b ☐ c3. ☐ a ☐ b ☒ c4. ☒ a ☐ b ☐ c5. ☐ a ☐ b ☒ c

Las bases de datos objeto-relacionales

Objetivos

1. Tomar perspectiva histórica sobre la incidencia del paradigma de la orientación a objetos en la evolución del desarrollo de *software*.
2. Comprender los elementos básicos de la orientación a objetos.
3. Familiarizarse con las características de las bases de datos objeto-relacionales.
4. Utilizar adecuadamente los *arrays* y tipos complejos.
5. Trabajar con referencias.
6. Aplicar correctamente el concepto de herencia.
7. Declarar y definir métodos.

Mapa conceptual del capítulo



Glosario

Camino más corto. Algoritmo ideado por E. Dijkstra para encontrar la ruta óptima dentro de un grafo. Es la base del protocolo OSPF utilizado en encaminamiento entre *routers*.

Crisis del *software*. Término utilizado para describir el estado del mundo del desarrollo de aplicaciones en la época previa al advenimiento de la ingeniería del *software*.

Desarrollo estructurado. Metodología de desarrollo de *software* imperante desde finales de los años sesenta. Favorece el uso controlado del flujo de instrucciones de programación, así como la creación de subrutinas para modularizar el código fuente.

LISP (*LISt Processing, procesamiento de listas*). Lenguaje de programación de listas utilizado en inteligencia artificial.

8.1. Introducción

En los años noventa el paradigma de la orientación a objetos apareció como alternativa a la metodología de desarrollo estructurado imperante hasta entonces. Empresas y organismos gubernamentales empezaron a acometer proyectos de desarrollo de *software* aplicando técnicas de análisis, diseño y programación orientada a objetos. El mundo de las bases de datos no quedó al margen, y en esa época surgieron varios sistemas gestores de bases de datos orientados a objetos (SGBDOO) que estructuraban la información almacenada en la base de datos en un formato idóneo para su gestión posterior por parte de un lenguaje de programación orientado a objetos. Además, dichas bases de datos permitían almacenar información compleja con menos restricciones de las exigidas por el modelo relacional.

Actualmente la cuota de mercado de los SGBDOO es más discreta de lo esperado, pero los fabricantes de soluciones relacionales vieron con buenos ojos la incorporación de ciertos conceptos de orientación a objetos en sus SGBDR. Hoy en día algunos de los gestores más importantes (Oracle, Microsoft SQL Server, IBM DB2, PostgreSQL, SAP ASE, IBM Informix) pertenecen a la clasificación de sistema gestor de base de datos objeto-relacionales (SGBDOR). Dado que el uso de elementos de orientación a objetos no es obligatorio, se pueden seguir utilizando como SGBDR tradicionales.

PARA SABER MÁS



Los primeros sistemas comerciales de bases de datos objeto-relacionales surgieron a finales de los años noventa. Bajo el nombre de *servidores universales (universal servers)*, su objetivo era combinar elementos del modelo relacional y del modelo orientado a objetos. Entre esos sistemas se encontraban IBM DB2 Universal Database, Informix Universal Data Option y Oracle Universal Server.

8.1.1. El paradigma de la orientación a objetos

La orientación a objetos busca mejorar la calidad y disminuir el coste de diseño y desarrollo de *software*. Presente en numerosos proyectos tanto de forma independiente como en combinación con la metodología estructurada, la dificultad inherente al proceso de análisis y su falta de flexibilidad ante imprevistos ha impedido que revolucionase el mercado del desarrollo como se preveía hace veinte años. Conceptualmente consiste en una representación más directa del mundo real, observando cada elemento a tratar como un objeto, definiendo en *clases* las características que deben cumplir objetos similares, creando *atributos* para cada dato de un objeto y escribiendo *métodos* a modo de funciones y procedimientos que llevan a cabo las acciones que se pueden realizar sobre dichos objetos.

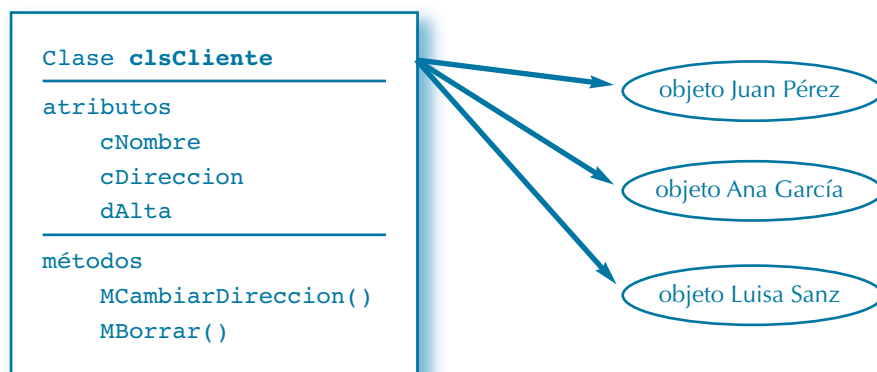


Figura 8.1. Ejemplo de clase con tres objetos

En el ejemplo de la figura 8.1 observamos la definición de una clase `clsCliente`, cuyos objetos cuentan con los atributos `cNombre`, `cDireccion` y `dAlta`, y los métodos `MCambiarDireccion()` y `MBorrar()`. Si quisiéramos eliminar al cliente Juan Pérez no ejecutaríamos una sentencia de borrado, ni llamaríamos a un procedimiento almacenado, sino al método `MBorrar()`, que se ejecuta exclusivamente sobre objetos de la clase `clsCliente`.

La orientación a objetos favorece la modularidad del código fuente. Algunas de sus características principales son las siguientes:

- *Encapsulación.* Los objetos solo pueden ser manejados a través de la interfaz que definen sus atributos y métodos, sin necesidad de conocer cómo están implementados internamente estos últimos.
- *Reutilización.* Gracias a la encapsulación, las clases son fácilmente exportables a otros sistemas, ya que para utilizarlas no es necesario conocer cómo están implementadas por dentro.
- *Herencia.* Mecanismo mediante el que una clase (*subclase* o *clase hija*) adquiere automáticamente los atributos y métodos de otra clase (*superclase* o *clase padre*), especializándose a partir de esta.

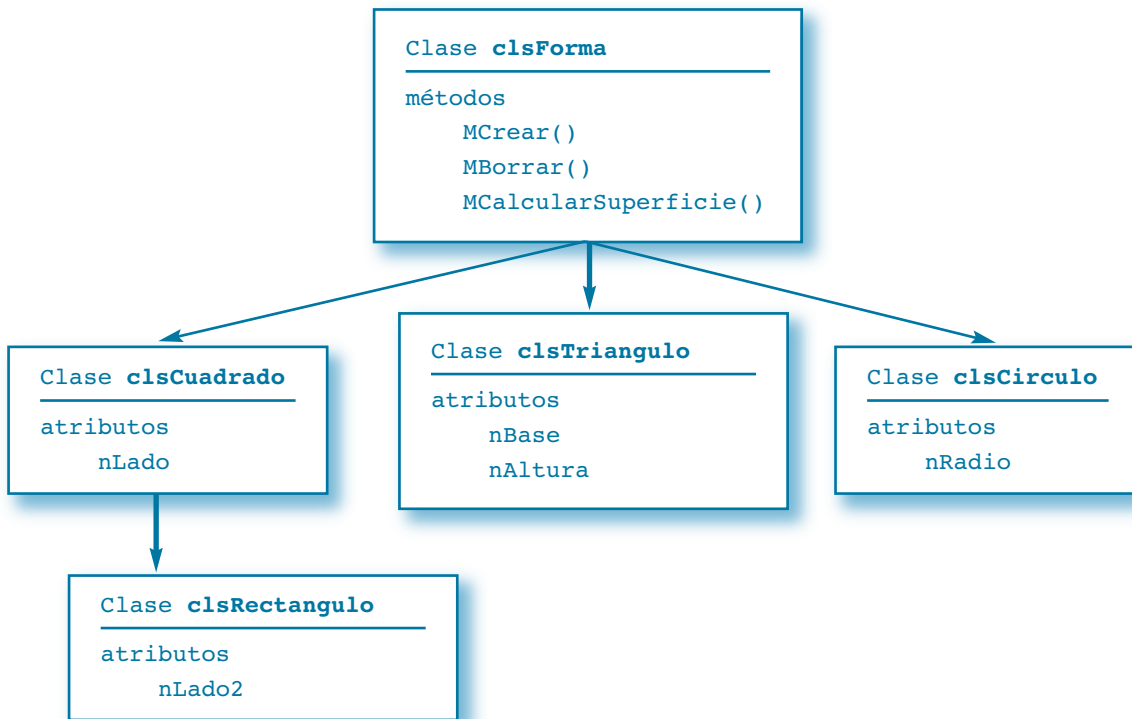
**Figura 8.2**

Diagrama de objetos en el que se muestra un ejemplo de herencia. La clase `clsRectangulo` hereda el atributo `nLado` de su superclase `clsCuadrado`, y las cuatro subclases heredan los tres métodos de la superclase `clsForma`.

- *Polimorfismo.* Derivada de la herencia, esta característica implica un comportamiento distinto del mismo método dependiendo de a qué clase pertenezca el objeto sobre el que se aplica. Continuando con el ejemplo de la figura 8.2, tanto los objetos de la clase `clsCuadrado` como los de `clsTriangulo` o `clsCirculo` podrán ejecutar el método `MCalcularSuperficie()`, ya que lo han heredado de la clase `clsForma`. No obstante, el comportamiento interno de dicho método será distinto para cada uno de los objetos, ya que estará reescrito en cada una de las clases, correspondiendo a $nLado^2$ para `clsCuadrado`, $nBase \times nAltura / 2$ para `clsTriangulo` y $\pi \times nRadio^2$ para `clsCirculo`.

PARA SABER MÁS

La orientación a objetos como alternativa a la metodología estructurada ha recibido fuertes críticas por parte de distintas eminencias. El experto en lenguaje LISP Paul Graham dijo que “la frase ‘orientación a objetos’ quiere decir muchas cosas. La mitad son obvias, y la otra mitad son errores”. Rob Pike, miembro del equipo del sistema operativo Unix, comentaba que “los diseños orientados a objetos son los números romanos de la informática”. El holandés Edsger W. Dijkstra, autor del término “crisis del *software*” y creador del algoritmo para resolver el problema del camino más corto (algoritmo de Dijkstra), entre otros logros, decía que “la programación orientada a objetos es una idea excepcionalmente mala que solo podía haber surgido en California”.



Actividad propuesta 8.1

Escribir un posible diseño orientado a objetos para la relación de jerarquía de la figura 2.23. Conceptualizar las posibles clases y sus atributos, y proponer métodos que puedan resultar útiles.

8.1.2. Características de las bases de datos objeto-relacionales

Aunque los distintos sistemas comerciales han ampliado considerablemente el repertorio objeto-relacional, el estándar SQL:1999 ya incluía sus elementos básicos, en los que nos centraremos en este capítulo. De acuerdo con la teoría de la orientación a objetos, en un modelo objeto-relacional cada registro de una tabla se considera un objeto, y la definición de la tabla su clase.

La novedad más importante de las bases de datos objeto-relacionales es su capacidad de gestionar tipos de datos complejos definidos por el usuario, con severas consecuencias para el modelo relacional subyacente:

- Se puede incluir varios valores en un mismo campo, con lo que se contradice frontalmente la primera forma normal, al no garantizar la atomicidad de los atributos.

nLibroID	cTitulo	acTemas
14	Computer Networks	{Informática, Redes}
17	The Tempest	{Teatro, Literatura clásica, Poscolonialismo}
...		

Figura 8.3. Tabla de libros con varios valores en el campo acTemas

- Como los tipos de datos pueden representar una estructura, las tablas dejan de ser elementos bidimensionales, y pueden incluir nuevas dimensiones de profundidad de la información.

nLibroID	cTitulo	oEditorial
14	Computer Networks	{cNombre: Prentice-Hall cPais: Estados Unidos}
17	The Tempest	{cNombre: Oxford cPais: Reino Unido}
...		

Figura 8.4. Tabla de libros con campo oEditorial complejo

Dado que el modelo objeto-relacional permite que los valores de un campo sean registros de otra tabla (que, a su vez, podría contar con registros de una tercera tabla como valores de sus propios campos), esta forma de almacenar la información se conoce como *modelo relacional anidado*. Gracias a esta característica se pueden eliminar tablas de relación y simplificar las sentencias **SELECT** de consulta.



Figura 8.5. Anidamiento entre tablas

Como se verá más adelante, las bases de datos objeto-relacionales también contemplan la herencia y los métodos.

8.2. Tipos de datos objeto

A continuación se verá la sintaxis de los dos grandes tipos de datos objeto-relacionales contemplados en el estándar SQL: los *arrays* y los tipos complejos.

8.2.1. Colecciones (*arrays*)

En programación un *array* (traducido a veces como “arreglo”) es una lista de elementos a los que se puede acceder mediante su posición en la lista. Por ese motivo, y dado que un *array* permite asignar varios valores al mismo campo, conceptualmente se conoce a los campos de tipo *array* como *atributos multivaluados*.

La creación de campos *array* implica una ligera alteración sobre la sintaxis SQL relacional:

```
CREATE TABLE TLibro(
    nLibroID IDENTITY PRIMARY KEY,
    cTitulo VARCHAR(40) NOT NULL,
    acTemas VARCHAR(30) ARRAY[10]
);
```

Figura 8.6

Sentencia de creación de una tabla con un campo tipo *array*. El campo *acTemas* (nótese que se ha prefijado la “c” de *character* con una “a” de *array*) puede albergar hasta diez valores VARCHAR(30)

```
INSERT INTO TLibro(cTitulo, acTemas)
VALUES ("The Tempest",
        ARRAY("Teatro", "Literatura clásica", "Poscolonialismo"));
```

Figura 8.7

Sentencia de inserción en una tabla con un campo tipo *array*. No es necesario rellenar los diez valores posibles del campo *acTemas*

```
SELECT cTitulo, acTemas[1], acTemas[2]
FROM TLibro
WHERE nLibroID = 32;
```

cTitulo	acTemas[1]	acTemas[2]
The Tempest	Teatro	Literatura clásica

Figura 8.8

Consulta sobre una tabla con un campo tipo *array* y salida de la consulta. Se puede acceder a cada valor del *array* mediante su número de posición

Actividad propuesta 8.2



Partiendo de la base de datos representada en la figura 3.28, identificar qué campos de qué tablas son candidatos a ser definidos mediante un tipo *array*. Razonar la respuesta.

8.2.2. Tipos complejos

Como se vio en la figura 8.4, el modelo objeto-relacional contempla la creación de nuevos tipos de datos estructurados por parte del usuario. La definición de dichos tipos puede servir para definir un campo de una tabla o bien la tabla completa, como se ve en el siguiente ejemplo:

```
CREATE TYPE typEditorial AS (  
    cNombre VARCHAR(30),  
    cPais VARCHAR(30)  
);  
CREATE TYPE typLibro AS (  
    cTitulo VARCHAR(40),  
    ...  
    oEditorial typEditorial  
);  
CREATE TABLE TLibro OF TYPE typLibro;
```

```
INSERT INTO TLibro  
VALUES ("The Tempest",  
    ...  
    typEditorial("Oxford", "Reino Unido")  
);
```

Figura 8.9
Sentencia de creación de la tabla TLibro a partir del tipo complejo typLibro que, a su vez, cuenta con un campo oEditorial del tipo complejo typEditorial; y sentencia de inserción en dicha tabla

```
SELECT cTitulo, oEditorial.cNombre
FROM TLibro
WHERE nLibroID = 32;
```

cTitulo	oEditorial.cNombre
-----	-----
The Tempest	Oxford

Figura 8.10

Consulta sobre una tabla con un campo de tipo complejo y salida de la consulta. Se puede acceder a cada valor del tipo complejo calificándolo con el nombre del campo al que pertenece

El usuario también puede definir los *arrays* como tipos:

```
CREATE TYPE typTemas AS (
    VARCHAR(30) ARRAY[10]
);
CREATE TYPE typLibro AS (
    cTitulo VARCHAR(40),
    ...
    acTemas typTemas,
    oEditorial typEditorial
);
CREATE TABLE TLibro OF TYPE typLibro;
```

Figura 8.11

Sentencia de creación del tipo *typTemas* basado en un *array* e inclusión de dicho tipo en el tipo *typLibro*, del que posteriormente derivará la tabla *TLibro*

También es posible definir un tipo ROW, específico para definir los campos de un registro. Un objeto de tipo ROW puede ser un valor válido para un campo, se puede almacenar en una variable de un bloque de código SQL/PSM, o bien puede funcionar como parámetro o valor devuelto por una subrutina:

```
CREATE TYPE typLibro AS (  
    cTitulo VARCHAR(40),  
    ...  
    oEditorial ROW(cNombre VARCHAR(30),  
                   cPais VARCHAR(30))  
);  
CREATE TABLE TLibro OF TYPE typLibro;
```

```
INSERT INTO TLibro  
VALUES ("The Tempest",  
    ...  
    ROW("Oxford", "Reino Unido")  
);
```

Figura 8.12

Creación de un campo de tipo ROW e inserción en dicho campo. La consulta SELECT sobre este tipo de campos es análoga a la que se realizaría sobre un tipo complejo

PARA SABER MÁS

Los tipos de datos objeto aparecieron en SQL:1999 a raíz de su inclusión en diversos gestores comerciales. Estos implementaron otros tipos de datos nuevos no incluidos en el estándar hasta su versión SQL:2003, como el tipo *conjunto* (*set*, similar al array, pero sin limitación de número de valores) y el *multiconjunto* (*multiset*, un tipo conjunto desordenado donde se permiten valores repetidos).

Actividad propuesta 8.3

Partiendo de la base de datos representada en la figura 3.28, identificar qué campos o grupos de campos de qué tablas son candidatos a ser definidos mediante un tipo complejo. Razonar la respuesta.



8.3. Referencias e identificadores

El modelo objeto-relacional no solo permite definir tipos estructurados; también es posible que los valores de dichos tipos estén almacenados como registros en tablas independientes, de modo que se obtengan tres ventajas:

1. El valor de un campo no reside físicamente en la tabla a la que pertenece dicho campo.
2. Como ese valor corresponde a un registro en otra tabla, varias tablas pueden usarlo como valor de un campo propio, eliminando redundancias.
3. Por ese mismo motivo, también se puede trabajar con los valores de los campos de dicho registro directamente en la tabla a la que pertenece.

Esta forma de gestionar el almacenamiento permite que una tabla guarde una *referencia* a la información de un campo. Dicha información se almacena físicamente en otra tabla distinta:

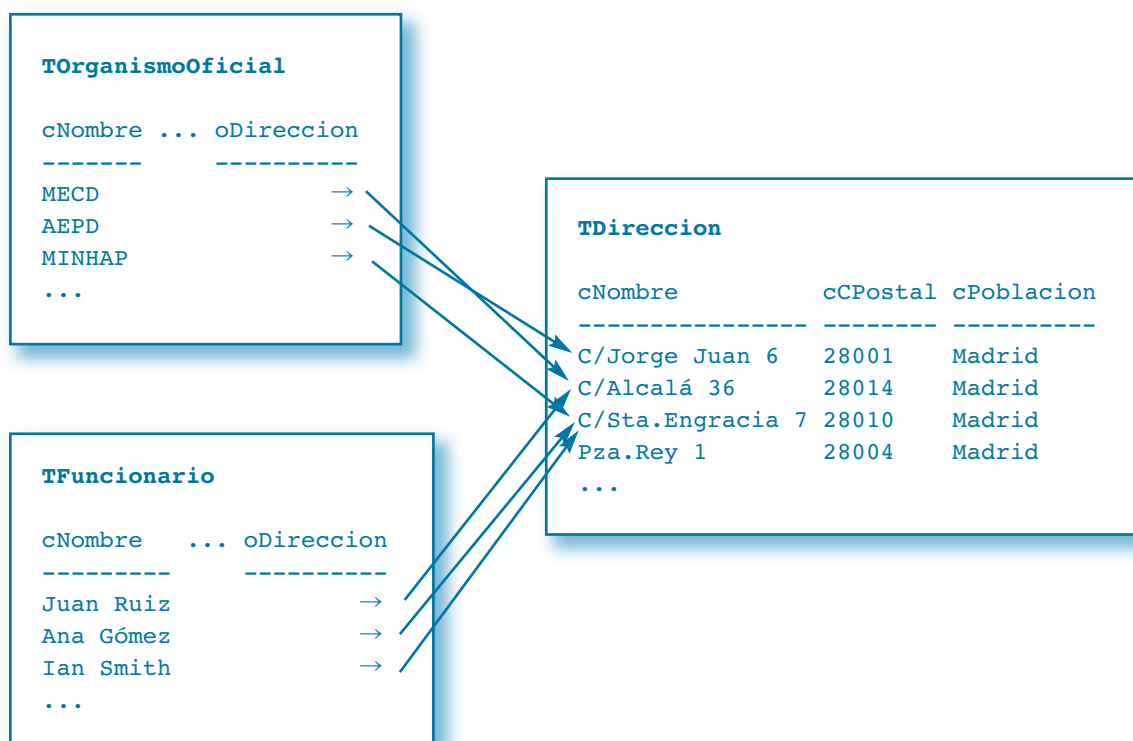


Figura 8.13

Ejemplo de referencia. Las direcciones de organismos oficiales y las direcciones de trabajo de funcionarios no se almacenan en sus respectivas tablas, sino en la tabla **TDireccion**. Dos direcciones son referenciadas por **TOrganismoOficial** y **TFuncionario** (una de ellas por dos registros distintos de **TFuncionario**). Otra solo es referenciada desde **TOrganismoOficial**. Otra no cuenta con referencias externas

Como se puede observar, el campo `oDireccion` funciona como una especie de clave ajena. Las siguientes sentencias especifican cómo crear esta estructura:

```
CREATE TYPE typDireccion (
    cNombre VARCHAR(120),
    cCPostal CHAR(5),
    cPoblacion VARCHAR(40)
);
CREATE TABLE TDireccion OF TYPE typDireccion;
```

```
CREATE TYPE typFuncionario (
    cNombre VARCHAR(90),
    ...
    oDireccion REF(typDireccion) SCOPE TDireccion
);
CREATE TABLE TFuncionario OF TYPE typFuncionario;
```

Figura 8.14

Creación de la tabla `TDireccion` basada en el tipo `typDireccion` y de la tabla `TFuncionario` de tipo `typFuncionario`, que referencia registros de `TDireccion`.

`REF` indica el tipo de datos del campo y `SCOPE` su ámbito, es decir, la estructura física que se va referenciar (en este caso la tabla `TDireccion`)

La inserción y actualización de datos cuando se usan referencias es análoga a la vista en la figura 8.9 con el uso de tipos complejos. La consulta, sin embargo, exige el uso de un nuevo operador, el *puntero* (`->`). Gracias a él estaremos “apuntando” (referenciando) al valor que buscamos en la tabla en que reside:

```
SELECT cNombre, oDireccion->cNombre, oDireccion->cPoblacion
FROM TFuncionario;
```

cNombre	oDireccion->cNombre	oDireccion->cPoblacion
Juan Ruiz	C/Alcalá 36	Madrid
Ana Gómez	C/Sta.Engracia 7	Madrid

Figura 8.15

Consulta de información con referencia a otra tabla, y salida de la consulta

Un error habitual consiste en especificar información por referencia utilizando un punto en vez de un puntero. En ese caso, lo que se obtendrá es una cadena de caracteres sin sentido, ya que SQL devolverá el *identificador* mediante el que el lenguaje sabe cómo llegar al dato apuntado:

```
SELECT cNombre, oDireccion.cNombre, oDireccion.cPoblacion
FROM TFuncionario;
```

cNombre	oDireccion.cNombre	oDireccion.cPoblacion
Juan Ruiz	29A53229JEK73F81	E03101402C40BBF3
Ana Gómez	28R22123JRR89U90	E03100220A87II02

Figura 8.16

Consulta errónea de información con referencia a otra tabla



Actividad propuesta 8.4

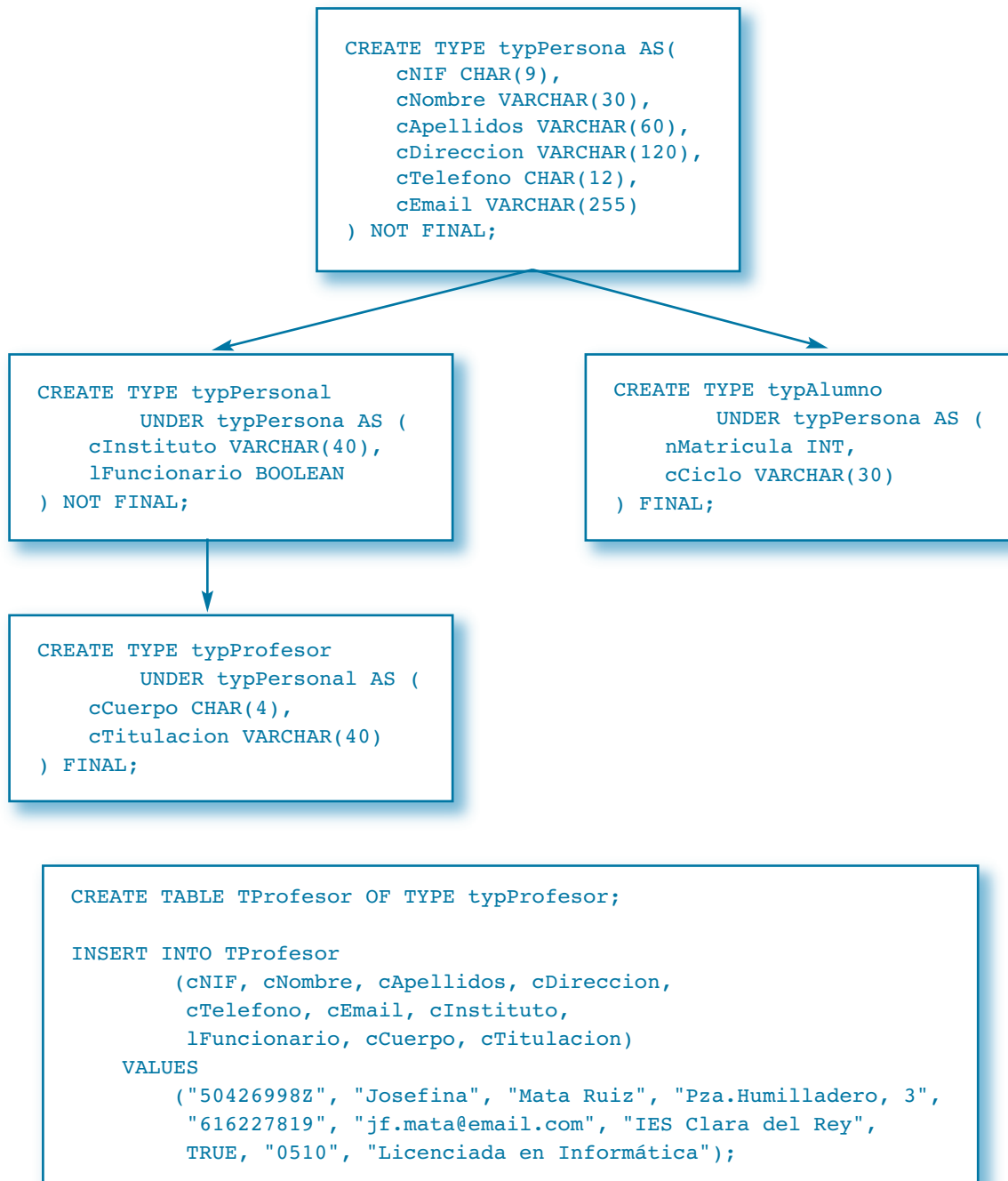
Partiendo de la base de datos representada en la figura 3.28, identificar en qué ocasiones el diseño podría beneficiarse del uso de referencias. Razonar la respuesta.

8.4. Herencia

Los mecanismos de herencia de las bases de datos objeto-relacionales permiten aplicar dicha característica a distintos objetos. Es por eso que hablamos de dos tipos de herencia:

8.4.1. De tipos

Un tipo (*subtipo*) puede heredar todos los atributos y métodos de otro tipo (*supertipo*) mediante la cláusula **UNDER**. Especificar **NOT FINAL** al final de la declaración permite que haya otros subtipos derivados (de modo que el tipo creado sería subtipo de un supertipo pero, a la vez, podría ser supertipo de otros subtipos). **FINAL** prohíbe crear más subtipos.

**Figura 8.17**

Creación de un supertipo `typPersona` con dos subtipos, uno de ellos supertipo, a su vez, de otro; y sentencias de creación de una tabla bajo el último subtipo e inserción de un registro. Nótese que el subtipo `typProfesor` hereda los atributos tanto de su supertipo `typPersonal` como del supertipo de este `typPersona`

PARA SABER MÁS



Diversos SGBDOR permiten la *herencia múltiple*, es decir, la posibilidad de que un subtipo pueda heredar los atributos de más de un supertipo. De ese modo un subtipo de *typPersona* (figura 8.17) también podría serlo de *typDireccion* (figura 8.14).



Actividad propuesta 8.5

Diseñar un esquema de herencia de tipos que represente la siguiente estructura jerárquica:

Animales	(Mamíferos	(Perros, monos),
	Aves	(Palomas, gaviotas),
	Peces	(Truchas, barbos, doradas))

Asignar atributos a cada tipo y escribir las sentencias de creación de todos ellos.

8.4.2. De tablas

La herencia de tablas permite implantar de forma efectiva los conceptos de especialización y generalización vistos en el diagrama entidad/relación extendido bajo el epígrafe de *jerarquías*. No solo se heredan los atributos, sino también los registros, de modo que una tabla (*tabla padre*) de la que heredan otras (*tablas hijas*) puede contener registros propios y registros correspondientes a tablas hijas, como se ve en las siguientes figuras:

```
CREATE TABLE TDireccion OF TYPE typDireccion;

CREATE TYPE typEmpresa UNDER typDireccion (
    cRazonSocial VARCHAR(40)
);
CREATE TABLE TCliente OF TYPE typEmpresa
    UNDER TDireccion;

CREATE TABLE TProveedor OF TYPE typEmpresa
    UNDER TDireccion;
```

Figura 8.18. Sentencias de creación de la tabla *TDireccion* a partir del tipo *typDireccion* de la figura 8.14, creación del tipo *typEmpresa* y de dos tablas de tipo *typEmpresa* hijas de *TDireccion*

TCliente				
cRazonSocial	cNombre	cCPostal	...	
Gª Hnos.	C/Puerto Rico 32	28016	...	
...				

TProveedor				
cRazonSocial	cNombre	cCPostal	...	
Ferrecol	C/Almagro 10	28010	...	
...				

TDireccion				
cNombre	cCPostal	...		
C/Almagro 10	28010	...		
C/Puerto Rico 32	28016	...		
Avda.Mediterráneo 42	28007	...		
...				

Figura 8.19

Datos de las tres tablas. TDireccion (tabla padre) tiene un registro correspondiente a TCliente (tabla hija), otro correspondiente a TProveedor (tabla hija) y otro independiente

Actividad propuesta 8.6

Partiendo del resultado de la actividad propuesta 8.1, diseñar tablas que representen la estructura jerárquica de la figura 2.23 y escribir las sentencias de creación de dichas tablas.



8.5. Métodos

Como se ha explicado en el apartado 8.1.1, en el mundo de la orientación a objetos los bloques de código se ejecutan en forma de métodos asociados a los objetos de cada clase, en vez de invocar a funciones y procedimientos independientes. Todo método, al igual que los atributos, forma parte de la definición de clase. En un contexto objeto-relacional, los métodos van asociados a la definición de los tipos de datos, como se ve en el siguiente ejemplo:

```
CREATE TYPE typFactura AS (
    nFactura INT,
    cCliente VARCHAR(60),
    dFactura DATE,
    nImporte NUMERIC(8,2)
) METHOD MAPlicarIVA(pnIVA REAL);
```

Figura 8.20
Declaración del tipo typFactura especificando sus atributos y un método (MAPlicarIVA())

Los métodos operan sobre los atributos de cada registro. Como ocurre en programación orientada a objetos con los objetos de toda clase, para referirse a dicho registro se utiliza la palabra `SELF`.

```
CREATE METHOD MAplicarIVA(pnIVA REAL)
    FOR typFactura
BEGIN
    SET SELF.nImporte = SELF.nImporte + ((SELF.nImporte / 100) * pnIVA);
END;
```

Figura 8.21. Código fuente del método `MAplicarIVA()`

Todo tipo estructurado, *array* o *ROW* cuenta con una función especial llamada *constructora*. La función constructora sirve para inicializar los atributos de un registro y devolverlo ya creado. Debe tener el mismo nombre que el tipo. Dado que se trata de una función implícita, no es necesario codificarla, excepto si se quiere añadir algún tipo de restricción o comprobación especial. Cada vez que se cree un nuevo registro de un tipo la función constructora se ejecutará automáticamente.

```
INSERT INTO TLibro
    VALUES ("The Tempest",
        ...
        typEditorial("Oxford", "Reino Unido")
    );
```

Figura 8.22

Inserción de un registro en una tabla (tomado de la figura 8.9). La creación del valor del campo de tipo `typEditorial` está invocando implícitamente a la función constructora del tipo

De acuerdo con la idea de encapsulación vista en el apartado 8.1.1, la interfaz de todo registro debería gestionarse exclusivamente a través de métodos, cuyo código albergaría las instrucciones SQL/PSM correspondientes.

Como las bases de datos objeto-relacionales permiten el polimorfismo, el mismo método puede reescribirse para distintos tipos:

```
CREATE METHOD MActualizarNombre(pcNombre VARCHAR(120))  
  FOR typDireccion  
BEGIN  
  IF SELF.cNombre <> pcNombre THEN  
    SET SELF.cNombre = pcNombre;  
  END IF;  
END;
```

```
CREATE METHOD MActualizarNombre(pcRazonSocial VARCHAR(40))  
  FOR typEmpresa  
BEGIN  
  IF SELF.cRazonSocial <> pcRazonSocial THEN  
    SET SELF.cRazonSocial = pcRazonSocial;  
  END IF;  
END;
```

Figura 8.23. Ejemplo de polimorfismo. El método MActualizarNombre() aparece escrito para el supertipo typDireccion y reescrito para su subtipo typEmpresa. Nótese que el nombre y tipo de datos del parámetro difiere en ambos métodos. También podrían recibir un número de parámetros distinto

Actividad propuesta 8.7

Escribir un método MDuplicar() para el tipo typFactura de la figura 8.20 que duplique el importe de la factura que se esté tratando si se emitió en agosto de 2014.



Resumen

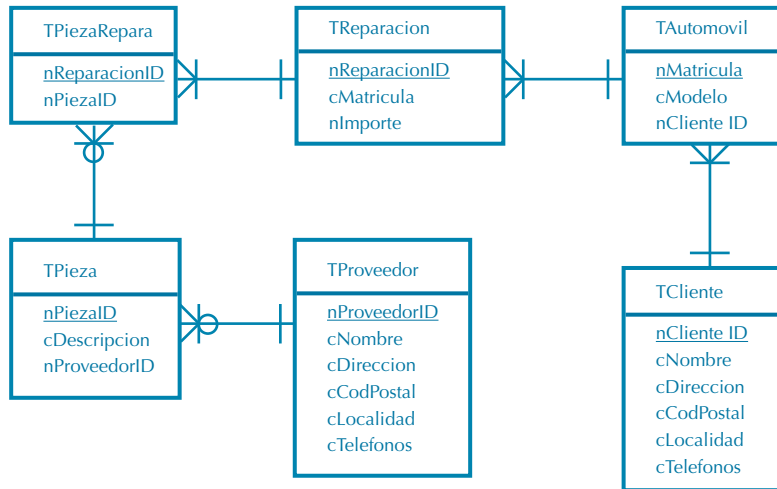
Una de las consecuencias de la irrupción del paradigma de la orientación a objetos en los años noventa fue la creación de bases de datos orientadas a objetos. Posteriormente diversos fabricantes de SGBDR incorporaron características de la orientación a objetos a sus productos relaciones, dando salida a las nuevas bases de datos objeto-relacionales.

En los SGBDOR los registros de las tablas se tratan como objetos, los campos como sus atributos y las funciones y procedimientos asociadas como métodos. El elemento más importante del nuevo modelo es la existencia de colecciones (*arrays*) y tipos estructurados cuya definición corresponde al usuario. Aunque trastocan considerablemente la base teórica del modelo relacional e incumplen, en el primer caso, la primera forma normal, ofrecen una flexibilidad de la que el modelo relacional carecía.

Las bases de datos objeto-relacionales permiten la implantación de métodos con polimorfismo, la herencia de tipos y de tablas y las referencias a registros de otras tablas, mediante las que reducen redundancias y complejidad de diseño.

EJERCICIOS PROPUESTOS

1. Considérese el siguiente modelo físico de datos correspondiente a un taller mecánico:



Rediseñarlo de acuerdo a las posibilidades del modelo objeto-relacional teniendo en cuenta, como mínimo, los siguientes preceptos:

- a) TProveedor y TCliente deben derivar de un mismo tipo.
 - b) Dicho tipo debe heredar atributos de un tipo typDireccion que albergue los datos de la dirección.
 - c) La lista de teléfonos de proveedor y cliente debe tratarse como un *array*.
 - d) La tabla TPiezaRepara no será necesaria, ya que las piezas aparecerán en TReparacion por referencia.
2. Escribir un método que permita cambiar una dirección.
 3. Escribir un método que permita añadir una pieza a una reparación en curso.
 4. Escribir un método que asigne a una variable vcDireccion la dirección completa de un cliente a partir de la matrícula de uno de sus coches.
 5. Buscar en Internet información sobre las prestaciones objeto-relacionales de los SGBDOR más populares.

ACTIVIDADES DE AUTOEVALUACIÓN

1. El paradigma de la orientación a objetos se sustenta en el uso de:
 - ☐ a) Tipos, objetos y atributos.
 - ☐ b) Clases, objetos, tipos y registros.
 - ☐ c) Clases, objetos, atributos y métodos.
2. La declaración de tipos *array*:
 - ☐ a) Contradice la primera forma normal.
 - ☐ b) Contradice las tres primeras formas normales.
 - ☐ c) No contradice la teoría de la normalización.
3. El uso de referencias:
 - ☐ a) Crea duplicidades.
 - ☐ b) Permite que valores de campos residan físicamente en otra tabla.
 - ☐ c) Copia valores de campos entre varias tablas.
4. Un supertipo:
 - ☐ a) Solo puede serlo de un subtipo.
 - ☐ b) Puede ser, a su vez, subtipo de otro tipo.
 - ☐ c) Debe ser subtipo de otro tipo.
5. Los métodos:
 - ☐ a) Forman parte de la definición de los tipos.
 - ☐ b) Llevan a cabo las mismas tareas que las funciones.
 - ☐ c) Están obsoletos.

SOLUCIONES:

1. ☐ a ☐ b ☒ c2. ☒ a ☐ b ☐ c3. ☐ a ☒ b ☐ c4. ☐ a ☒ b ☐ c5. ☒ a ☐ b ☐ c

Bibliografía

Publicaciones

- Abad, A. (2013): *Seguridad y alta disponibilidad*. Garceta. Madrid.
- ANSI/ISO/IEC (1999): *Database Language SQL – Part 2: Foundation (SQL/Foundation) “Part 2”*. ANSI/ISO/IEC.
- Cabrera, G. (2001): *Sistemas gestores de bases de datos*. Paraninfo. Madrid.
- Celko, J. (2005): *SQL for Smarties. Advanced SQL Programming*, 3ª ed. Elsevier Inc. San Francisco (California).
- Centro de Estudios Financieros (2004): *Gestión de sistemas e informática de la Administración del Estado*. Centro de Estudios Financieros. Madrid.
- Chamberlin, D., y Boyce, R. (1974): *SEQUEL: A Structured English Query Language*. IBM Research Laboratory. San José (California).
- Codd, E. F. (1970): “A Relational Model of Data for Large Shared Data Banks”, en *Communications of the ACM*, vol. 13, nº 6, P. Baxendale.
- (1990): *The Relational Model for Database Management. Version 2*. Addison-Wesley.
- Cuadra, D., y otros (2007): *Desarrollo de bases de datos: casos prácticos desde el análisis a la implementación*. Ra-Ma. Madrid.
- Date, C. J. (2006): *Date on Database. Writings 2000-2006*. Apress. Berkeley (California).
- (2012): *SQL and Relational Theory. How to Write Accurate SQL Code*. O’Reilly. Sebastopol (California).
- Garcia-Molina, H.; Ullman, J. y Widom, J. (2002): *Database Systems: The Complete Book*. Prentice Hall. New Jersey.
- González, A. (2010): *Gestión de bases de datos*. Ra-Ma. Madrid.

- Gregorio, S., y otros (2002): *Análisis y diseño detallado de aplicaciones informáticas de gestión. Guía práctica de técnicas*. Mira Editores. Zaragoza.
- Harrington, J. (2010): *SQL Clearly Explained*, 3ª ed. Elsevier Inc. Burlington (Massachusetts).
- Haztefuncionario.com (2004): *Cuerpo de Gestión de Informática*. haztefuncionario.com. Madrid
- Kline, K., y Kline, D. (2001): *SQL in a Nutshell*. O'Reilly.
- Kriegel, A., y Trukhnov, B. (2008): *SQL Bible*. Wiley Publishing, Inc. Indianapolis (Indiana).
- Laiho, M., y Laux, F. (2014): «Introduction to Procedural Extensions of SQL» (borrador). DBTechNet.org.
- López, I.; Castellano, M. J. y Ospino, J. (2011): *Bases de datos*. Garceta. Madrid.
- Melton, J. (ed.) (2001): *WG3:SXF-002. DM32.2-2012-00005. Information Technology – Database languages – SQL – Part 1: Framework (SQL/Framework)*. ANSI.
- Miguel, A. de, y Piattini, M. (1993): *Concepción y diseño de bases de datos relacionales. Del modelo E/R al modelo relacional*. Ra-Ma. Madrid.
- Miguel, A. de,; Piattini, M., y Marcos, E. (2000): *Diseño de bases de datos relacionales*. Ra-Ma. Madrid.
- Ministerio de Administraciones Públicas (2001): *MÉTRICA Versión 3*. Ministerio de Administraciones Públicas. Madrid.
- Oppel, A. (2009): *Databases: A Beginner's Guide*. McGraw-Hill.
- Piattini, M., y otros (2004): *Análisis y diseño de aplicaciones informáticas de gestión. Una perspectiva de ingeniería del software*. Ra-Ma. Madrid.
- Pressman, R. (2002): *Software Engineering. A Practitioner's Approach. European Adaptation*, 5ª ed. McGraw-Hill. Nueva York. Traducción española de Ojeda, R., y otros: *Ingeniería del software. Un enfoque práctico*, 5ª ed. McGraw-Hill/Interamericana de España. Madrid.
- Silberschatz, A; Korth, H., y Sudarshan, S. (2002): *Database Systems Concepts*, 4ª ed. McGraw-Hill. Nueva York. Traducción española de Sáenz, F., y otros: *Fundamentos de bases de datos*, 4ª ed. McGraw-Hill/Interamericana de España. Madrid.
- Song, I.; Evans, M., y Park, E. K. (1995): «A Comparative Analysis of Entity-Relationship Diagrams», en *Journal of Computer and Software Engineering*, vol. 3, n° 4, pp. 427-459. Ablex Publishing. Norwood (Nueva Jersey).
- Taylor, A. (2011): *SQL All-in-One For Dummies*, 2ª ed. Wiley Publishing Inc. Hoboken (Nueva Jersey).
- Urman, S. (1998): *Oracle8, PL/SQL Programming*. Oracle Press. Traducción española: *Oracle8. Programación PL/SQL*. McGraw-Hill. Madrid.

Páginas web

developer.mimer.se
 dictionary.cambridge.org
 en.wikipedia.org
 hsqldb.org
 ibm.com
 microsoft.com
 mysql.com
 odbms.org
 onlamp.com

oracle.com
postgresql.org
rae.es
sap.com
sybase.com
techrepublic.com
teradata.com

Gráficos e imágenes

Figuras 6.1, 6.2, 7.1 y 7.2 generadas con CISCO Stencils.
Imágenes tomadas de www.flickr.com:

- Reproducidas bajo licencia Creative Commons BY 2.0 (<https://creativecommons.org/licenses/by/2.0/>):
 - Imagen de *mainframe* (figura 1.1) cortesía de Pargon.
 - Imagen de interfaz de base de datos (figura 1.10) cortesía de Linux Screenshots.
 - Imagen de Charles Simonyi (figura 3.10) cortesía de Marcin Wichary.
- Reproducidas sin modificaciones bajo licencia Creative Commons BY SA 2.0 (<https://creativecommons.org/licenses/by-sa/2.0/>):
 - Imagen de discos RAID (figura 7.4) cortesía de John Athayde.
 - Imagen de puerta de CPD (figura 7.6) cortesía de Mack Male.
 - Imagen de *backup* (figura 7.7) cortesía de Björn Söderqvist.
 - Imagen de *backup* (figura 7.8) cortesía de Luigi Rosa.

