

Sistemas de Gestión de Bases de datos

Tercera edición



Mc
Graw
Hill

Ramakrishnan • Gehrke

SISTEMAS DE GESTIÓN DE BASES DE DATOS

Tercera Edición

SISTEMAS DE GESTIÓN DE BASES DE DATOS

Tercera Edición

Raghuram Krishnan
*Universidad de Wisconsin
Madison, Wisconsin, EE.UU.*

Johannes Gehrke
*Universidad de Cornell
Ithaca, Nueva York, EE.UU.*

Traducción

Jesús Correas Fernández
Antonio García Cordero
Carolina López Martínez

Revisión técnica

Fernando Sáenz Pérez
Universidad Complutense de Madrid



MADRID BOGOTÁ BUENOS AIRES CARACAS GUATEMALA LISBOA MÉXICO
NUEVA YORK PANAMÁ SAN JUAN SANTIAGO SAO PAULO
AUCKLAND HAMBURGO LONDRES MILÁN MONTREAL NUEVA DELHI
PARÍS SAN FRANCISCO SIDNEY SINGAPUR ST. LOUIS TOKIO TORONTO

La información contenida en este libro procede de una obra original publicada por McGraw-Hill. No obstante, McGraw-Hill/Interamericana de España no garantiza la exactitud o perfección de la información publicada. Tampoco asume ningún tipo de garantía sobre los contenidos y las opiniones vertidas en dichos textos.

Este trabajo se publica con el reconocimiento expreso de que se está proporcionando una información, pero no tratando de prestar ningún tipo de servicio profesional o técnico. Los procedimientos y la información que se presentan en este libro tienen sólo la intención de servir como guía general.

McGraw-Hill ha solicitado los permisos oportunos para la realización y el desarrollo de esta obra.

Sistemas de gestión de bases de datos, Tercera Edición

No está permitida la reproducción total o parcial de este libro, ni su tratamiento informático, ni la transmisión de ninguna forma o por cualquier medio, ya sea electrónico, mecánico, por fotocopia, por registro u otros métodos, sin el permiso previo y por escrito de los titulares del Copyright.



**McGraw-Hill/Interamericana
de España, S.A.U.**

DERECHOS RESERVADOS © 2007, respecto a la tercera edición en español, por

McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S.A.U.
Edificio Valrealty, 1^a planta
Basauri, 17
28023 Aravaca (Madrid)

*http://www.mcgraw-hill.es
universidad@mcgraw-hill.com*

Traducido de la tercera edición en inglés de
Database Management Systems
ISBN: 0-07-246563-8

Copyright 2003 por The McGraw-Hill Companies, Inc.

**ISBN: 978-84-481-5638-1
Depósito legal: M.**

Editor: Carmelo Sánchez González
Técnico editorial: Israel Sebastián
Compuesto por: Fernando Sáenz Pérez
Impreso en:

IMPRESO EN ESPAÑA — PRINTED IN SPAIN

Para Apu, Ketan y Vivek con amor.

Para Keiko y Elisa.

CONTENIDO

PREFACIO	xix
Parte I FUNDAMENTOS	1
1 INTRODUCCIÓN A LOS SISTEMAS DE BASES DE DATOS	3
1.1 Gestión de datos	4
1.2 Perspectiva histórica	5
1.3 Sistemas de archivos y SGBD	7
1.4 Ventajas de los SGBD	8
1.5 Descripción y almacenamiento de datos en los SGBD	10
1.5.1 El modelo relacional	10
1.5.2 Niveles de abstracción en los SGBD	12
1.5.3 Independencia con respecto a los datos	14
1.6 Las consultas en los SGBD	15
1.7 Gestión de transacciones	16
1.7.1 Ejecución concurrente de las transacciones	16
1.7.2 Las transacciones no completadas y los fallos del sistema	17
1.7.3 Puntos a destacar	18
1.8 Arquitectura de los SGBD	18
1.9 Usuarios de las bases de datos	20
1.10 Preguntas de repaso	21
2 INTRODUCCIÓN AL DISEÑO DE BASES DE DATOS	25
2.1 Diseño de bases de datos y diagramas ER	26
2.1.1 Más allá del diseño ER	27
2.2 Entidades, atributos y conjuntos de entidades	28
2.3 Las relaciones y los conjuntos de relaciones	29
2.4 Otras características del modelo ER	32
2.4.1 Restricciones de clave en relaciones	32
2.4.2 Restricciones de participación	34
2.4.3 Entidades débiles	35
2.4.4 Jerarquías de clases	37

2.4.5	Agregación	39
2.5	Diseño conceptual del modelo ER	41
2.5.1	Entidades y atributos	41
2.5.2	Entidades y relaciones	42
2.5.3	Relaciones binarias y ternarias	43
2.5.4	Agregación y relaciones ternarias	45
2.6	Diseño conceptual para grandes empresas	46
2.7	El lenguaje unificado de modelado	47
2.8	Estudio de un caso: la tienda en Internet	49
2.8.1	Análisis de requisitos	49
2.8.2	Diseño conceptual	49
2.9	Preguntas de repaso	50
3	EL MODELO RELACIONAL	57
3.1	Introducción al modelo relacional	59
3.1.1	Creación y modificación de relaciones mediante SQL	61
3.2	Restricciones de integridad sobre relaciones	63
3.2.1	Restricciones de clave	64
3.2.2	Restricciones de clave externa	66
3.2.3	Restricciones generales	67
3.3	Cumplimiento de las restricciones de integridad	68
3.3.1	Transacciones y restricciones	71
3.4	Consultas de datos relacionales	72
3.5	Diseño lógico de bases de datos: del modelo ER al relacional	73
3.5.1	De los conjuntos de entidades a las tablas	73
3.5.2	De los conjuntos de relaciones (sin restricciones) a las tablas	74
3.5.3	Traducción de conjuntos de relaciones con restricciones de clave	76
3.5.4	Traducción de los conjuntos de relaciones con restricciones de participación	78
3.5.5	Traducción de los conjuntos de entidades débiles	80
3.5.6	Traducción de las jerarquías de clase	81
3.5.7	Traducción de los diagramas ER con agregación	82
3.5.8	Del modelo ER al relacional: más ejemplos	83
3.6	Introducción a las vistas	84
3.6.1	Vistas, independencia de datos y seguridad	85
3.6.2	Actualizaciones de las vistas	85
3.7	Eliminación y modificación de tablas y vistas	89
3.8	Estudio de un caso: la tienda en Internet	89
3.9	Preguntas de repaso	91

4	ÁLGEBRA Y CÁLCULO RELACIONALES	97
4.1	Preliminares	98
4.2	Álgebra relacional	98
4.2.1	Selección y proyección	99
4.2.2	Operaciones con conjuntos	101
4.2.3	Renombramiento	103
4.2.4	Reuniones (<i>join</i>)	103
4.2.5	División	105
4.2.6	Más ejemplos de consultas algebraicas	106
4.3	Cálculo relacional	112
4.3.1	Cálculo relacional de tuplas	112
4.3.2	Cálculo relacional de dominios	117
4.4	Potencia expresiva del álgebra y del cálculo	119
4.5	Preguntas de repaso	120
5	SQL: CONSULTAS, RESTRICCIONES Y DISPARADORES	125
5.1	Introducción	126
5.1.1	Organización del capítulo	127
5.2	Forma de las consultas SQL básicas	128
5.2.1	Ejemplos de consultas básicas de SQL	132
5.2.2	Expresiones y cadenas de caracteres en la orden SELECT	133
5.3	UNION, INTERSECT y EXCEPT	135
5.4	Consultas anidadas	138
5.4.1	Introducción a las consultas anidadas	138
5.4.2	Consultas anidadas correlacionadas	140
5.4.3	Operadores para la comparación de conjuntos	141
5.4.4	Más ejemplos de consultas anidadas	142
5.5	Operadores de agregación	143
5.5.1	Las cláusulas GROUP BY y HAVING	146
5.5.2	Más ejemplos de consultas de agregación	150
5.6	Valores nulos	153
5.6.1	Comparaciones que emplean valores nulos	154
5.6.2	Las conectivas lógicas AND, OR y NOT	154
5.6.3	Consecuencias para las estructuras de SQL	154
5.6.4	Reuniones externas	155
5.6.5	Desactivación de los valores nulos	156
5.7	Restricciones de integridad complejas en SQL	156
5.7.1	Restricciones sobre una sola tabla	156

x	Sistemas de gestión de bases de datos	
	5.7.2 Restricciones de dominio y tipos distintos	157
	5.7.3 Asertos: RI para varias tablas	158
5.8	Disparadores y bases de datos activas	159
	5.8.1 Ejemplos de disparadores en SQL	160
5.9	Diseño de bases de datos activas	161
	5.9.1 Por qué los disparadores pueden resultar difíciles de comprender?	162
	5.9.2 Restricciones y disparadores	162
	5.9.3 Otros usos de los disparadores	163
5.10	Preguntas de repaso	163
Parte II	DESARROLLO DE APLICACIONES	171
6	DESARROLLO DE APLICACIONES DE BASE DE DATOS	173
6.1	Acceso a bases de datos desde aplicaciones	174
	6.1.1 SQL incorporado	175
	6.1.2 Cursor	177
	6.1.3 SQL dinámico	180
6.2	Una introducción a JDBC	181
	6.2.1 Arquitectura	182
6.3	Clases e interfaces JDBC	183
	6.3.1 Gestión de controladores JDBC	184
	6.3.2 Conexiones	184
	6.3.3 Ejecución de sentencias SQL	186
	6.3.4 ResultSets	187
	6.3.5 Excepciones y avisos	188
	6.3.6 Obtención de los metadatos de una base de datos	189
6.4	SQLJ	190
	6.4.1 Escritura de código SQLJ	192
6.5	Procedimientos almacenados	194
	6.5.1 Creación de un procedimiento almacenado simple	194
	6.5.2 Llamada a procedimientos almacenados	195
	6.5.3 SQL/PSM	197
6.6	Caso de estudio: la librería en Internet	198
6.7	Preguntas de repaso	201
7	APLICACIONES DE INTERNET	205
7.1	Introducción	205
7.2	Conceptos de Internet	206
	7.2.1 Identificadores uniformes de recursos (URI)	206

7.2.2	El protocolo de transferencia de hipertexto (HTTP)	208
7.3	Documentos HTML	210
7.4	Documentos XML	212
7.4.1	Introducción a XML	213
7.4.2	DTD de XML	215
7.4.3	DTD de dominios específicos	217
7.5	La arquitectura de aplicaciones de tres capas	220
7.5.1	Arquitecturas de una capa y cliente-servidor	220
7.5.2	Arquitecturas de tres capas	222
7.5.3	Ventajas de la arquitectura de tres capas	224
7.6	La capa de presentación	225
7.6.1	Formularios HTML	225
7.6.2	JavaScript	228
7.6.3	Hojas de estilo	231
7.7	La capa intermedia	233
7.7.1	CGI: la interfaz de pasarela común	233
7.7.2	Servidores de aplicaciones	234
7.7.3	Servlets	236
7.7.4	Páginas de servidor de Java	238
7.7.5	Mantenimiento del estado	238
7.8	Caso de estudio: la librería en Internet	243
7.9	Preguntas de repaso	245
8	VISIÓN GENERAL DE LA GESTIÓN DE TRANSACCIONES	253
8.1	Las propiedades ACID	254
8.1.1	Consistencia y aislamiento	255
8.1.2	Atomicidad y durabilidad	255
8.2	Transacciones y planificaciones	256
8.3	Ejecución concurrente de transacciones	257
8.3.1	Motivación para la ejecución concurrente	257
8.3.2	Secuencialidad	258
8.3.3	Anomalías debidas a la ejecución entrelazada	259
8.3.4	Planificaciones que implican transacciones abortadas	262
8.4	Control de concurrencia basado en bloqueos	263
8.4.1	Bloqueo en dos fases estricto (B2F estricto)	263
8.4.2	Interbloqueos	264
8.5	Rendimiento del bloqueo	266
8.6	Soporte de transacciones en SQL	267
8.6.1	Creación y terminación de transacciones	267

8.6.2	Qué debería bloquearse?	269
8.6.3	Características de las transacciones en SQL	270
8.7	Preguntas de repaso	272
Parte III ALMACENAMIENTO E ÍNDICES		277
9	INTRODUCCIÓN AL ALMACENAMIENTO Y LOS ÍNDICES	279
9.1	Datos en almacenamiento externo	280
9.2	Organizaciones de archivo e indexación	281
9.2.1	ndices agrupados	282
9.2.2	ndices primario y secundario	283
9.3	Estructuras de datos de Índices	284
9.3.1	Indexación asociativa (<i>Hash</i>)	284
9.3.2	Indexación basada en árboles	285
9.4	Comparación entre las organizaciones de archivo	287
9.4.1	Modelo de coste	288
9.4.2	Archivos de montículo	289
9.4.3	Archivos ordenados	290
9.4.4	Archivos agrupados	291
9.4.5	Archivo de montículo con índice en árbol no agrupado	292
9.4.6	Archivo de montículo con índice asociativo no agrupado	293
9.4.7	Comparación de los costes de E/S	295
9.5	ndices y ajuste del rendimiento	296
9.5.1	Impacto de la carga de trabajo	296
9.5.2	Organización de índices agrupados	297
9.5.3	Claves de búsqueda compuestas	299
9.5.4	Especificación de índices en SQL:1999	302
9.6	Preguntas de repaso	303
10	ÍNDICES DE ÁRBOL	307
10.1	Introducción a los Índices de árbol	308
10.2	Método de acceso secuencial indexado (ISAM)	310
10.2.1	Páginas de desbordamiento, consideraciones de bloqueo	312
10.3	Árboles B+: una estructura de Índice dinámica	313
10.3.1	Formato de un nodo	315
10.4	Búsqueda	315
10.5	Inserción	317
10.6	Borrado	320
10.7	Duplicados	325

10.8	Árboles B+ en la práctica	326
10.8.1	Compresión de claves	326
10.8.2	Carga masiva de un árbol B+	328
10.8.3	El concepto de orden	331
10.8.4	El efecto de las inserciones y borrados sobre los idr	331
10.9	Preguntas de repaso	332
11	INDEXACIÓN BASADA EN ASOCIACIÓN	339
11.1	Asociación estática	340
11.1.1	Notación y convenios	342
11.2	Asociación extensible	342
11.3	Asociación lineal	347
11.4	Asociación extensible frente a lineal	353
11.5	Preguntas de repaso	353
Parte IV DISEÑO Y AJUSTE DE BASES DE DATOS		359
12	REFINAMIENTO DE ESQUEMAS Y FORMAS NORMALES	361
12.1	Introducción al refinamiento de esquemas	362
12.1.1	Problema causados por la redundancia	362
12.1.2	Descomposiciones	364
12.1.3	Problemas relacionados con la descomposición	365
12.2	Dependencias funcionales	366
12.3	Razonamiento sobre las DF	367
12.3.1	Cierre de un conjunto de DF	367
12.3.2	Cierre de los atributos	369
12.4	Formas normales	369
12.4.1	La forma normal de Boyce-Codd	370
12.4.2	Tercera forma normal	371
12.5	Propiedades de las descomposiciones	373
12.5.1	Descomposición por reunión sin pérdida	373
12.5.2	Descomposiciones que conservan las dependencias	375
12.6	Normalización	376
12.6.1	Descomposición en FNBC	376
12.6.2	Descomposición en 3FN	378
12.7	Refinamiento de esquemas en el diseño de bases de datos	382
12.7.1	Restricciones en los conjuntos de entidades	382
12.7.2	Restricciones en los conjuntos de relaciones	383
12.7.3	Identificación de los atributos de las entidades	383

12.7.4	Identificación de los conjuntos de entidades	385
12.8	Otros tipos de dependencias	386
12.8.1	Dependencias multivaloradas	386
12.8.2	Cuarta forma normal	388
12.8.3	Dependencias de reunión	390
12.8.4	Quinta forma normal	390
12.8.5	Dependencias de inclusión	390
12.9	Estudio de un caso: la tienda en Internet	391
12.10	Preguntas de repaso	392
13	DISEÑO FÍSICO Y AJUSTE DE BASES DE DATOS	401
13.1	Introducción al diseño físico de bases de datos	402
13.1.1	Cargas de trabajo en las bases de datos	403
13.1.2	Diseño físico y decisiones de ajuste	404
13.1.3	Necesidad del ajuste de las bases de datos	405
13.2	Directrices para la selección de Índices	405
13.3	Ejemplos básicos de selección de Índices	407
13.4	Agrupación e indexación	409
13.4.1	Coagrupación de dos relaciones	411
13.5	Índices que permiten planes exclusivamente para Índices	413
13.6	Herramientas para la selección de Índices	414
13.6.1	Selección automática de índices	414
13.6.2	Funcionamiento de los asistentes para el ajuste de índices	415
13.7	Visión general del ajuste de bases de datos	418
13.7.1	Ajuste de índices	418
13.7.2	Ajuste del esquema conceptual	419
13.7.3	Ajuste de consultas y de vistas	420
13.8	Opciones de ajuste del esquema conceptual	421
13.8.1	Aceptación de formas normales más débiles	422
13.8.2	Desnormalización	422
13.8.3	Elección de la descomposición	423
13.8.4	Particiones verticales de las relaciones en FNBC	424
13.8.5	Descomposición horizontal	424
13.9	Opciones de ajuste de consultas y de vistas	425
13.10	Consecuencias de la concurrencia	427
13.10.1	Reducción de la duración de los bloqueos	427
13.10.2	Reducción de los puntos calientes	428
13.11	Estudio de un caso: la tienda en Internet	430
13.11.1	Ajuste de la base de datos	431

13.12	Pruebas de rendimiento de los SGBD	431
13.12.1	Pruebas de rendimiento de SGBD bien conocidas	432
13.12.2	Empleo de pruebas de rendimiento	433
13.13	Preguntas de repaso	434
14	SEGURIDAD Y AUTORIZACIÓN	441
14.1	Introducción a la seguridad de las bases de datos	442
14.2	Control de acceso	443
14.3	Control discrecional de acceso	444
14.3.1	Concesión y revocación de vistas y restricciones de integridad	451
14.4	Control obligatorio de acceso	453
14.4.1	Relaciones multinivel y poliinstanciación	454
14.4.2	Canales ocultos, niveles de seguridad del DoD	455
14.5	Seguridad para las aplicaciones de Internet	456
14.5.1	Cifrado	457
14.5.2	Servidores de certificación: el protocolo SSL	459
14.5.3	Firmas digitales	460
14.6	Otros aspectos de la seguridad	461
14.6.1	Papel de los administradores de bases de datos	461
14.6.2	Seguridad en las bases de datos estadísticas	462
14.7	Estudio de un caso de diseño: la tienda en Internet	463
14.8	Preguntas de repaso	465
Parte V	TEMAS ADICIONALES	469
15	SISTEMAS DE BASES DE DATOS DE OBJETOS	471
15.1	Ejemplo motivador	473
15.1.1	Nuevos tipos de datos	474
15.1.2	Manipulación de los nuevos datos	475
15.2	Tipos de datos estructurados	478
15.2.1	Tipos colección	478
15.3	Operaciones con datos estructurados	479
15.3.1	Operaciones con filas	479
15.3.2	Operaciones con arrays	479
15.3.3	Operaciones con otros tipos colección	480
15.3.4	Consultas a colecciones anidadas	481
15.4	Encapsulación y TAD	482
15.4.1	Definición de métodos	483
15.5	Herencia	485

15.5.1	Definición de tipos con herencia	485
15.5.2	Métodos de vinculación	486
15.5.3	Jerarquías de colecciones	486
15.6	Objetos, IDO y tipos referencia	487
15.6.1	Conceptos de igualdad	488
15.6.2	Desreferencia de los tipos referencia	488
15.6.3	URL e IDO en SQL:1999	489
15.7	Diseño de bases de datos para SGBDROO	489
15.7.1	Tipos colección y TAD	489
15.7.2	Identidad de los objetos	492
15.7.3	Ampliación del modelo ER	493
15.7.4	Empleo de colecciones anidadas	494
15.8	Desafíos en la implementación de los SGBDROO	495
15.8.1	Almacenamiento y métodos de acceso	496
15.8.2	Procesamiento de consultas	497
15.8.3	Optimización de consultas	499
15.9	SGBDOO	501
15.9.1	El modelo de datos ODMG y ODL	501
15.9.2	OQL	503
15.10	Comparación entre SGBDR, SGBDOO y SGBDROO	505
15.10.1	SGBDR y SGBDROO	505
15.10.2	SGBDOO y SGBDROO: similitudes	505
15.10.3	SGBDOO y SGBDROO: diferencias	505
15.11	Preguntas de repaso	506
16	ALMACENES DE DATOS Y AYUDA A LA TOMA DE DECISIONES	513
16.1	Introducción a la ayuda a la toma de decisiones	515
16.2	OLAP: Modelo multidimensional de datos	516
16.2.1	Diseño de bases de datos multidimensionales	519
16.3	Consultas de agregación multidimensionales	520
16.3.1	ROLLUP y CUBE en SQL:1999	522
16.4	Consultas ventana en SQL:1999	525
16.4.1	Enmarcado de ventanas	527
16.4.2	Nuevas funciones de agregación	527
16.5	Búsqueda rápida de respuestas	528
16.5.1	Las consultas de los N primeros	528
16.5.2	Agregación en línea	529
16.6	Técnicas de implementación para OLAP	531

16.6.1	Índices de mapas de bits	531
16.6.2	Índices de reunión	533
16.6.3	Organizaciones de archivos	534
16.7	Almacenes de datos	535
16.7.1	Creación y mantenimiento de almacenes de datos	535
16.8	Vistas y ayuda a la toma de decisiones	537
16.8.1	Vistas, OLAP y almacenamiento	537
16.8.2	Consultas sobre vistas	537
16.9	Materialización de vistas	538
16.9.1	Problemas de la materialización de vistas	538
16.10	Mantenimiento de vistas materializadas	540
16.10.1	Mantenimiento incremental de vistas	541
16.10.2	Mantenimiento de las vistas en almacenes de datos	543
16.10.3	Sincronización de las vistas	545
16.11	Preguntas de repaso	545
17	MINERÍA DE DATOS	553
17.1	Introducción a la minería de datos	554
17.1.1	El proceso del descubrimiento de conocimiento	555
17.2	Recuento de apariciones conjuntas	556
17.2.1	Conjuntos de artículos frecuentes	556
17.2.2	Consultas iceberg	559
17.3	Minería de reglas	560
17.3.1	Reglas de asociación	560
17.3.2	Algoritmo para la búsqueda de reglas de asociación	561
17.3.3	Reglas de asociación y jerarquías ES	562
17.3.4	Reglas de asociación generalizadas	563
17.3.5	Pautas secuenciales	564
17.3.6	Empleo de las reglas de asociación para la predicción	565
17.3.7	Redes bayesianas	566
17.3.8	Reglas de clasificación y de regresión	566
17.4	Reglas estructuradas en árboles	568
17.4.1	Árboles de decisión	569
17.4.2	Un algoritmo para la creación de árboles de decisión	571
17.5	Agrupación	572
17.5.1	Un algoritmo de agrupación	574
17.6	Búsquedas de similitudes en las secuencias	575
17.6.1	Un algoritmo para hallar secuencias similares	577
17.7	Minería incremental y corrientes de datos	577

17.7.1	Mantenimiento incremental de lotes frecuentes	579
17.8	Otras tareas en la minería de datos	581
17.9	Preguntas de repaso	581
18	RECUPERACIÓN DE INFORMACIÓN Y DATOS XML	587
18.1	Mundos en conflicto: bases de datos, RI y XML	588
18.1.1	SGBD y sistemas de RI	589
18.2	Introducción a la recuperación de información	590
18.2.1	El modelo del espacio vectorial	591
18.2.2	Peso de los términos según la FT/FID	591
18.2.3	Clasificación de documentos por su similitud	593
18.2.4	Medida del éxito: precisión y recuperación	594
18.3	Indexación para la búsqueda de texto	594
18.3.1	Índices invertidos	595
18.3.2	Archivos de firmas	597
18.4	Motores de búsqueda Web	599
18.4.1	Arquitectura de los motores de búsqueda	599
18.4.2	Empleo de la información sobre los vínculos	600
18.5	Gestión del texto en SGBD	603
18.5.1	Índices invertidos débilmente acoplados	604
18.6	Modelo de datos para XML	605
18.6.1	Razón de la estructura laxa	605
18.6.2	Modelo de grafos	606
18.7	XQuery: Consulta de datos XML	607
18.7.1	Expresiones de ruta	607
18.7.2	Expresiones FLWR	608
18.7.3	Ordenación de los elementos	609
18.7.4	Agrupación y generación de colecciones	610
18.8	Evaluación eficiente de consultas XML	610
18.8.1	Almacenamiento de XML en SGBDR	611
18.8.2	Indexación de repositorios XML	614
18.9	Preguntas de repaso	617
BIBLIOGRAFÍA		625
ÍNDICE		647

PREFACIO

La ventaja de la autoalabanza es que se puede hacer tan grande como se desee y justo en el lugar adecuado.

— Samuel Butler

Los sistemas gestores de bases de datos son hoy en día una herramienta indispensable para la administración de la información, y los cursos sobre los principios y la práctica de los sistemas de bases de datos forman actualmente parte integral de cualquier plan de estudios de informática. Este libro trata de los fundamentos de los modernos sistemas gestores de bases de datos, en especial los sistemas de bases de datos relacionales.

Este libro es una versión abreviada de la tercera edición de *Database Management Systems* y orientada a cursos de diseño de bases de datos, en el que se han omitido los aspectos avanzados y de implementación de los sistemas gestores de bases de datos que se pueden encontrar en la versión completa.

En este libro se ha intentado presentar el material en un estilo sencillo y claro, empleando un enfoque cuantitativo con muchos ejemplos detallados. Un extenso conjunto de ejercicios (para los cuales los profesores disponen de soluciones en Internet) acompaña cada capítulo y refuerza la capacidad de los alumnos para aplicar los conceptos a problemas reales.

El libro se puede utilizar con el software y las propuestas de programación adjuntos en dos tipos diferentes de cursos de introducción:

1. **Énfasis en las aplicaciones.** Un curso que trata de los principios de los sistemas de bases de datos y pone el énfasis en la manera en que se utilizan para el desarrollo de aplicaciones que hacen un empleo intensivo de los datos. Se han añadido a la tercera edición dos nuevos capítulos sobre el desarrollo de aplicaciones (uno sobre aplicaciones que se apoyan en las bases de datos y otro sobre Java y las arquitecturas de aplicaciones de Internet), y todo el libro se ha revisado y reorganizado ampliamente para dar apoyo a un curso así. El estudio de un caso particular a lo largo de todo el libro y gran cantidad de material en Internet (por ejemplo, código para consultas SQL y aplicaciones de Java, bases de datos en Internet y soluciones) facilitan la enseñanza de un curso centrado en las aplicaciones prácticas.
2. **Énfasis en los sistemas.** Un curso que tiene un gran énfasis en los sistemas y que da por supuesto que los alumnos tienen buenas dotes de programación en C y en C++. En este caso, el software adjunto Minibase se puede emplear como base para proyectos en los que se pida a los alumnos que lleven a cabo diversas partes de un SGBD relacional. En el texto se describen con suficiente detalle varios módulos centrales del software del proyecto (por ejemplo, los árboles B+, los índices de asociación y diversos métodos de reunión) como para permitir a los alumnos ponerlos en práctica, dadas las interfaces de clase (C++).

No cabe duda de que muchos profesores enseñarán un curso que se hallará entre estos dos extremos. La reestructuración de la tercera edición ofrece una organización muy modular que facilita ese tipo de cursos híbridos. El libro contiene también suficiente material como para dar apoyo a un par de cursos más avanzados.

Organización de la tercera edición

El libro está organizado en cuatro partes principales más un conjunto de temas avanzados, como puede verse en la Figura 0.1. Los capítulos sobre fundamentos introducen los sistemas

(1) Fundamentos	Ambos
(2) Desarrollo de aplicaciones	Énfasis en las aplicaciones
(3) Almacenamiento e índices	Énfasis en los sistemas
(4) Diseño y ajuste de bases de datos	Énfasis en las aplicaciones
(5) Temas adicionales	Ambos

Figura 0.1 Organización de las partes de la tercera edición

de bases de datos, el modelo ER y el modelo relacional. Explican la manera en que se crean y se utilizan las bases de datos y tratan las bases del diseño y la consulta de bases de datos, incluido un tratamiento en profundidad de las consultas SQL. Aunque el profesor puede omitir parte de este material según su criterio (por ejemplo, el cálculo relacional, algunos apartados sobre el modelo ER o las consultas SQL), este material es importante para todos los alumnos de sistemas de bases de datos, y se recomienda tratarlo con todo el detalle posible.

Cada una de las cuatro partes restantes pone el énfasis en las aplicaciones o en los sistemas. La parte de sistemas tiene un capítulo introductorio, diseñado para ofrecer un tratamiento autocontenido. El Capítulo 8 es una introducción al almacenamiento y los índices. Cada capítulo introductorio se puede emplear para ofrecer un tratamiento independiente de cada tema, o bien como primer capítulo de un tratamiento más detallado. Por tanto, en un curso orientado a las aplicaciones, puede que el Capítulo 9 fuera el único material estudiado sobre organización de archivos e indexación, mientras que en un curso orientado a los sistemas estaría complementado con una selección de los Capítulos 10 al 11. La parte sobre diseño y ajuste de bases de datos contiene un estudio sobre el ajuste del rendimiento y del diseño de accesos seguros. Estos temas de aplicaciones se comprenden mejor tras haber dado a los alumnos una buena formación en la arquitectura de los sistemas de bases de datos y, por tanto, se colocan más avanzada la secuencia de capítulos.

Sugerencias de esquemas para cursos

Este libro se puede utilizar en dos tipos de cursos introductorios a las bases de datos, uno con el énfasis puesto en las aplicaciones y el otro con el énfasis puesto en los sistemas.

El *curso introductorio orientado a las aplicaciones* podría abarcar los capítulos sobre fundamentos, luego los capítulos de desarrollo de aplicaciones, seguidos por los capítulos de almacenamiento e índices y concluir con el material de diseño y ajuste de bases de datos. Se

han minimizado las dependencias entre capítulos, lo que permite a los profesores ajustar con facilidad el material que deseen incluir. El material sobre fundamentos (Parte I) debe tratarse en primer lugar y, en la Parte III se debe tratar en primer lugar el capítulo introductorio. Las únicas dependencias que sigue habiendo entre los capítulos de las Partes I a IV se muestran con flechas en la Figura 0.2. Conviene tratar en orden los capítulos de la Parte I. Sin embargo, se puede saltar el tratamiento del álgebra y del cálculo para llegar antes a las consultas de SQL (aunque los autores creen que este material es importante y recomiendan que se trate antes de SQL).

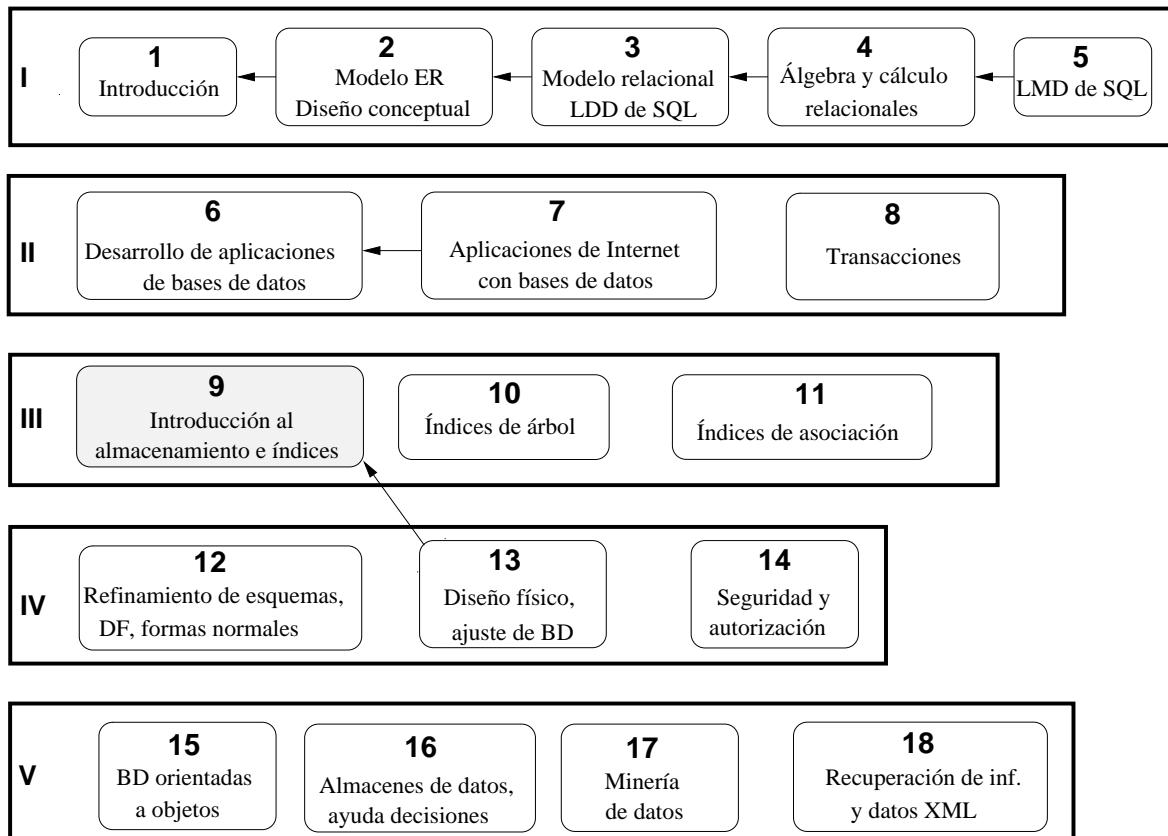


Figura 0.2 Organización y dependencias de los capítulos

El *curso introductorio orientado a los sistemas* abarcaría los capítulos de fundamentos y una selección de los capítulos de aplicaciones y de almacenamiento e índices. Un aspecto importante de los cursos orientados a sistemas es que el calendario de los proyectos de programación (por ejemplo, el empleo de Minibase) hace deseable que se traten al principio algunos temas de sistemas. Las dependencias entre capítulos se han limitado cuidadosamente para permitir que los capítulos sobre sistemas se den tan pronto se hayan dado los Capítulos 1 y 3. Los restantes capítulos de fundamentos y de aplicaciones se pueden dar posteriormente.

El libro también tiene amplio material para apoyar una secuencia de varios cursos. Evidentemente, la selección de un énfasis en aplicaciones o en sistemas para el curso introductorio

hace que se pase por alto cierto material en ese curso; el material del libro permite dar una sucesión de dos cursos que traten tanto los aspectos de las aplicaciones como de introducción a los sistemas. Los temas adicionales abarcan una amplia gama de asuntos, y se pueden utilizar como material principal de un curso avanzado complementado con lecturas adicionales.



Material complementario

Este libro tiene materiales suplementarios en Internet (en inglés):

<http://www.cs.wisc.edu/~dbbook>

- **Capítulo en Internet.** Para dejar sitio para el material nuevo, como el desarrollo de las aplicaciones, la recuperación de la información y XML, se ha trasladado el tratamiento de QBE a un capítulo en Internet. Los alumnos pueden descargarse libremente este capítulo del sitio Web del libro, y las soluciones a los ejercicios de ese capítulo están incluidas en el manual de soluciones.
- **Transparencias de las lecciones magistrales.** Las transparencias de las lecciones magistrales se encuentran a libre disposición de los lectores para todos los capítulos en formatos Postscript y PDF. Los profesores de los cursos también pueden obtenerlas en el formato Microsoft Powerpoint, y adaptarlas a sus necesidades docentes. Los profesores también tienen acceso a todas las figuras empleadas en el libro (en formato xfig), y pueden emplearlas para modificar las transparencias.
- **Soluciones de los ejercicios de cada capítulo.** Este libro tiene un conjunto excepcionalmente amplio de ejercicios de gran profundidad. Los alumnos pueden obtener en Internet las soluciones de los ejercicios impares y un conjunto de transparencias de lecciones magistrales de cada capítulo en los formatos Postscript y Adobe PDF. Los profesores de cada asignatura pueden obtener las soluciones de todos los ejercicios.
- **Software.** Este libro viene acompañado de dos tipos de software. En primer lugar, está Minibase, un SGBD relacional de pequeño tamaño pensado para su empleo en cursos orientados a los sistemas. Minibase viene con ejemplos y soluciones, como se describe en el Apéndice 30. El acceso queda restringido a los profesores de cada asignatura. En segundo lugar, se ofrece código para todos los ejercicios de desarrollo de aplicaciones en SQL y en Java del libro, junto con secuencias de comandos para la creación de bases de datos de ejemplo y otras para la configuración de varios SGBD comerciales. Los alumnos sólo pueden tener acceso al código de las soluciones de los ejercicios impares, mientras que los profesores tienen acceso a todas las soluciones.

Para obtener más información

La página Web de este libro se halla en el URL:

<http://www.cs.wisc.edu/~dbbook>

Contiene una lista de las modificaciones entre la segunda y la tercera edición y un *enlace* que se actualiza con frecuencia a *todas las erratas conocidas del libro y de sus suplementos*

adjuntos. Conviene que los profesores visiten este sitio periódicamente o que se registren en él para que se les notifiquen las modificaciones importantes por correo electrónico.

Agradecimientos

Este libro nació de las notas para las clases de CS564, el curso introductorio (nivel senior/graduado) de la Universidad de Wisconsin-Madison. David DeWitt desarrolló este curso y el proyecto Minirel, en el que los alumnos escribieron varias partes seleccionadas de un SGBD relacional. Las ideas del autor acerca de este material estaban influidas por su docencia de CS564, y Minirel fue la inspiración de Minibase, que es más general (por ejemplo, tiene un optimizador de consultas e incluye software de visualización) pero intenta conservar el espíritu de Minirel. Mike Carey y el autor diseñaron conjuntamente gran parte de Minibase. Las notas para las clases (y, luego, este libro) estaban influidas por las notas para las clases de Mike y por las transparencias para las clases de Yannis Ioannidis.

Joe Hellerstein empleó la edición beta de este libro en Berkeley y proporcionó una interacción valiosísima, ayuda con las transparencias e hilarantes citas. La escritura del capítulo sobre sistemas de bases de datos de objetos con Joe fue divertidísima.

C. Mohan ofreció una ayuda valiosísima, respondiendo pacientemente gran número de preguntas sobre técnicas de implementación empleadas en diferentes sistemas comerciales, en especial la indexación, el control de la concurrencia y los algoritmos de recuperación. Moshe Zloof respondió a numerosas preguntas sobre la semántica de QBE y sobre los sistemas comerciales basados en QBE. Ron Fagin, Krishna Kulkarni, Len Shapiro, Jim Melton, Dennis Shasha y Dirk Van Gucht repasaron el libro y ofrecieron una interacción detallada que mejoró enormemente su contenido y su presentación. Michael Goldweber de Beloit College, Matthew Haines de Wyoming, Michael Kifer de SUNY StonyBrook, Jeff Naughton de Wisconsin, Praveen Seshadri de Cornell y Stan Zdonik de Brown emplearon también la edición beta en sus cursos sobre bases de datos y proporcionaron interacción e informes sobre fallos. En concreto, Michael Kifer señaló un error en el (antiguo) algoritmo de cálculo de recubrimientos mínimos y sugirió el tratamiento de algunas características de SQL en el Capítulo 2 para mejorar la modularidad. La bibliografía de Gio Wiederhold, pasada a formato LaTex por S. Sudarshan, y la bibliografía en línea sobre bases de datos y programación lógica de Michael Ley resultaron de gran ayuda para la compilación de la bibliografía de cada capítulo. Shaun Flisakowski y Uri Shaft ayudaron frecuentemente al autor en sus inacabables batallas con LaTex.

Un agradecimiento especial a los muchísimos alumnos que han colaborado en el software de Minibase. Emmanuel Ackaouy, Jim Pruyne, Lee Schumacher y Michael Lee trabajaron con el autor cuando éste desarrollaba la primera versión de Minibase (gran parte de la cual se ha descartado posteriormente, pero tuvo influencia en la siguiente versión). Emmanuel Ackaouy y Bryan So eran los profesores ayudantes del autor cuando éste daba el curso CS564 empleando esa versión y superaron ampliamente los límites de una ayudantía en sus esfuerzos para mejorar el proyecto. Paul Aoki luchó con una versión de Minibase y ofreció gran cantidad de comentarios útiles como profesor ayudante en Berkeley. Todo un curso de alumnos de CS764 (el curso de bases de datos para graduados) desarrolló gran parte de la versión actual de Minibase en un gran proyecto de curso que fue dirigido y coordinado por Mike Carey.

y por el autor. Amit Shukla y Michael Lee eran los profesores ayudantes del autor cuando éste dio por primera vez el curso CS564 empleando esta versión de Minibase y continuaron desarrollando el software.

Varios alumnos trabajaron con el autor en proyectos independientes a lo largo de mucho tiempo para desarrollar componentes de Minibase. Entre éstos figuran los paquetes de visualización para el gestor de la memoria intermedia y los árboles B+ (Huseyin Bektas, Harry Stavropoulos y Weiqing Huang); un optimizador y visualizador de consultas (Stephen Harris, Michael Lee y Donko Donjerkovic); una herramienta para diagramas ER basada en el editor de esquemas Opossum (Eben Haber); y una herramienta de normalización con interfaz gráfica (Andrew Prock y Andy Therber). Además, Bill Kimmel trabajó en la integración y reparación de un gran fragmento de código (el gestor de almacenamiento, el gestor de la memoria intermedia, los archivos y los métodos de acceso, los operadores relacionales y el ejecutor de planes de consulta) generado por el proyecto del curso CS764. Ranjani Ramamurty amplió considerablemente el trabajo de Bill para la limpieza e integración de diferentes módulos. Luke Blanshard, Uri Shaft y Shaun Flisakowski trabajaron para montar la versión de distribución del código y desarrollaron conjuntos de exámenes y ejercicios basados en el software de Minibase. Krishna Kunchithapadam probó el optimizador y desarrolló parte de la interfaz gráfica de Minibase.

Evidentemente, el software Minibase no existiría sin las aportaciones de gran número de personas con talento. Con este software libremente disponible en el dominio público el autor espera que más profesores sean capaces de dar cursos de bases de datos orientados a sistemas con una mezcla de implementación y experimentación que complementen el material de clase.

El autor quisiera dar las gracias a los muchos alumnos que ayudaron a desarrollar y comprobar las soluciones a los ejercicios y ofrecieron una útil realimentación con los borradores de este libro. En orden alfabético: X. Bao, S. Biao, M. Chakrabarti, C. Chan, W. Chen, N. Cheung, D. Colwell, C. Fritz, V. Ganti, J. Gehrke, G. Glass, V. Gopalakrishnan, M. Higgins, T. Jasmin, M. Krishnaprasad, Y. Lin, C. Liu, M. Lusignan, H. Modi, S. Narayanan, D. Randolph, A. Ranganathan, J. Reminga, A. Therber, M. Thomas, Q. Wang, R. Wang, Z. Wang y J. Yuan. Arcady Grenader, James Harrington y Martin Reames de Wisconsin y Nina Tang de Berkeley proporcionaron una interacción especialmente detallada.

Charlie Fischer, Avi Silberschatz y Jeff Ullman dieron al autor valiosísimos consejos sobre el trabajo con los editores. Los editores de McGraw-Hill, Betsy Jones y Eric Munson llevaron a cabo amplias revisiones y guiaron este libro en sus primeras etapas. Emily Gray y Brad Kosirog estuvieron allí cuando surgieron problemas. En Wisconsin, Ginny Werner ayudó realmente al autor a mantener las cosas bajo control.

Finalmente, este libro robó al autor mucho tiempo y, en muchos aspectos, fue más duro para la familia del autor que para él mismo. Sus hijos se expresaron sin ambages. En palabras de su hijo de (entonces) cinco años, Ketan: “Papá, deja de trabajar en esa tontería de libro. No te queda tiempo para *mí*.” Vivek, de dos años: “¿Estás trabajando en el *libro*? ¡No, no, no, ven a jugar al baloncesto conmigo!” Todos sus momentos de descontento recayeron sobre la mujer del autor y, pese a todo, Apu mantuvo alegremente a la familia funcionando de la manera caótica y feliz habitual las muchísimas noches y fines de semana que el autor estuvo liado con este libro (¡por no mencionar los días en que estaba liado con el trabajo en la universidad!). Como en todas las cosas, el autor puede ver la mano de sus padres en gran parte de esto; su padre, con su amor por el aprendizaje, y su madre, con su amor por

la familia, lo educaron. Las contribuciones de su hermano Kartik a este libro consistieron principalmente en llamadas telefónicas que apartaban al autor de su trabajo, pero si no le diera las gracias puede que se molestara. El autor quisiera dar las gracias a su familia por estar ahí y dar sentido a todo lo que hace. (¡Mira por dónde! El autor sabía que encontraría un motivo legítimo para dar las gracias a Kartik.)

Agradecimientos de la segunda edición

Emily Gray y Betsy Jones de McGraw-Hill realizaron amplias revisiones y ofrecieron consejo y apoyo mientras los autores preparaban la segunda edición. Jonathan Goldstein ayudó con la bibliografía de bases de datos espaciales. Los siguientes revisores ofrecieron una valiosa interacción sobre el contenido y la organización del libro: Liming Cai de la Universidad de Ohio, Costas Tsatsoulis de la Universidad de Kansas, Kwok-Bun Yue de la Universidad de Houston, Clear Lake, William Grosky de la Universidad Wayne State, Sang H. Son de la Universidad de Virginia, James M. Slack de la Universidad Minnesota State, Mankato, Herman Balsters de la Universidad de Twente, Holanda, Karen C. Davis de la Universidad de Cincinnati, Joachim Hammer de la Universidad de Florida, Fred Petry de la Universidad Tulane, Gregory Speegle de la Universidad Baylor, Salih Yurttas de la Universidad Texas A&M y David Chao de la Universidad San Francisco State.

Gran número de personas comunicó errores de la primera edición. En concreto, los autores quisieran dar las gracias a las siguientes: Joseph Albert de la Universidad Portland State, Han-yin Chen de la Universidad de Wisconsin, Lois Delcambre del Instituto Oregon Graduate, Maggie Eich de la Universidad Southern Methodist, Raj Gopalan de la Universidad Tecnológica Curtin, Davood Rafiei de la Universidad de Toronto, Michael Schrefl de la Universidad de South Australia, Alex Thomasian de la Universidad de Connecticut y Scott Vandenberg de Siena College.

Un agradecimiento especial a las muchas personas que respondieron una encuesta detallada sobre el soporte de los sistemas comerciales a diversas características: en IBM, Mike Carey, Bruce Lindsay, C. Mohan y James Teng; en Informix, M. Muralikrishna y Michael Ubell; en Microsoft, David Campbell, Goetz Graefe y Peter Spiro; en Oracle, Hakan Jacobsson, Jonathan D. Klein, Muralidhar Krishnaprasad y M. Ziauddin; y en Sybase, Marc Chanliau, Lucien Dimino, Sangeeta Doraiswamy, Hanuma Kodavalla, Roger MacNicol y Tirumanjanam Rengarajan.

Tras verse en los agradecimientos de la primera edición, Ketan (ya con ocho años) planteó una pregunta sencilla: “¿Cómo es que no nos dedicaste el libro a nosotros? ¿Por qué a mamá?” Ketan, he corregido ese inexplicable descuido. Vivek (ahora con cinco años) estaba más preocupado por la amplitud de su fama: “Papá, ¿está mi nombre en *todas* las copias de tu libro? ¿Lo tienen en *todos* los departamentos de Informática del mundo?” Vivek, eso espero. Finalmente, esta revisión no habría salido a la luz sin el apoyo de Apu y de Keiko.

Agradecimientos de la tercera edición

Los autores quieren dar las gracias a Raghav Kaushik por su aportación al estudio de XML, y a Alex Thomasian por su aportación al tratamiento del control de la concurrencia. Un

agradecimiento especial a Jim Melton por darles una copia de su libro sobre extensiones orientadas a objetos de la norma SQL:1999 previa a la edición y por detectar varias erratas en un borrador de esta edición. Marti Hearst de Berkeley les permitió generosamente adaptar parte de sus transparencias sobre Recuperación de la Información y Alon Levy y Dan Suciu fueron lo bastante amables como para dejarles adaptar varias de sus clases magistrales sobre XML. Mike Carey ofreció información sobre los servicios Web.

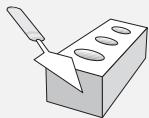
Emily Lupash de McGraw-Hill ha sido una fuente de apoyo y ánimos constante. Coordinó amplias revisiones por Ming Wang de la Universidad Aeronáutica Embry Riddle, Cheng Hsu de RPI, Paul Bergstein de la Universidad de Massachusetts, Archana Sathaye de SJSU, Bharat Bhargava de Purdue, John Fendrich de Bradley, Ahmet Ugur de Central Michigan, Richard Osborne de la Universidad de Colorado, Akira Kawaguchi de CCNY, Mark Last de Ben Gurion, Vassilis Tsotras de Univ. of California y Ronald Eaglin de la Universidad de Central Florida. Es un placer reconocer la meditada aportación recibida de los revisores, que mejoró enormemente el diseño y el contenido de esta edición. Gloria Schiesl y Jade Moran aguantaron con alegría y eficiencia situaciones caóticas de última hora y, con Sherry Kane, hicieron posible que se cumpliera un calendario muy ajustado. Michelle Whitaker modificó en numerosas ocasiones el diseño de la cubierta y de la contraportada.

En una nota personal para Raghu, Ketan, siguiendo el bello ejemplo del camello que compartía una tienda de campaña, observó que “tan sólo era justo” que Raghu le dedicara esta edición únicamente a él, ya que “mamá ya tenía una dedicada sólo a ella”. Pese a este descarado intento de acaparar la atención, apoyado con entusiasmo por Vivek y contemplado con el indulgente afecto de un padre que los adora, este libro también está dedicado a Apu, por estar siempre ahí.

En cuanto a Johannes, esta revisión no habría visto la luz sin el apoyo de Keiko y la inspiración y la motivación de ver la pacífica cara dormida de Elisa.

PARTE I

FUNDAMENTOS



1

INTRODUCCIÓN A LOS SISTEMAS DE BASES DE DATOS

- ➔ ¿Qué son los SGBD y, en concreto, los SGBD relacionales?
- ➔ ¿Por qué se debe pensar en un SGBD a la hora de gestionar datos?
- ➔ ¿Cómo se representan los datos de las aplicaciones en los SGBD?
- ➔ ¿Cómo se recuperan y se tratan los datos en los SGBD?
- ➔ ¿Cómo soportan los SGBD el acceso concurrente y cómo protegen los datos durante los fallos del sistema?
- ➔ ¿Cuáles son los componentes principales de un SGBD?
- ➔ ¿Quién tiene relación con las bases de datos en la vida real?
- ➔ **Conceptos fundamentales:** administración de bases de datos, independencia con respecto a los datos, diseño de bases de datos, modelos de datos; bases de datos relacionales y consultas; esquemas, niveles de abstracción; transacciones, concurrencia y bloqueos, recuperación y registros; arquitectura de los SGBD; administradores de bases de datos, programadores de aplicaciones, usuarios finales.

¿Se han dado cuenta todos de que todas las letras de la palabra *database* (base de datos en inglés) se escriben a máquina con la mano izquierda? Ahora bien, la disposición del teclado de máquina de escribir QWERTY se diseñó, entre otras cosas, para facilitar un uso equilibrado de ambas manos. Se deduce, por tanto, que escribir sobre bases de datos no sólo es antinatural, sino también mucho más difícil de lo que parece.

— Anónimo

La cantidad de información disponible crece, literalmente, de manera explosiva, y el valor de los datos como activo de las organizaciones está ampliamente reconocido. Para que los usuarios obtengan el máximo rendimiento de sus enormes y complejos conjuntos de datos son necesarias herramientas que simplifiquen las tareas de administrar los datos y de extraer

4 Sistemas de gestión de bases de datos

El campo de los sistemas gestores de bases de datos es un microcosmos dentro de la informática en general. Los aspectos abordados y las técnicas empleadas abarcan un amplio espectro, que incluye los lenguajes, la orientación a objetos y otros paradigmas de la programación, la compilación, los sistemas operativos, la programación concurrente, las estructuras de datos, los algoritmos, la teoría, los sistemas paralelos y distribuidos, las interfaces de usuario, los sistemas expertos y la inteligencia artificial, las técnicas estadísticas y la programación dinámica. No es posible tratar todos los aspectos de la gestión de bases de datos en un solo libro, pero los autores esperan transmitir al lector una idea de la excitación que provoca esta rica y vibrante disciplina.

información útil en el momento preciso. En caso contrario, los datos pueden convertirse en una carga cuyo coste de adquisición y de gestión supere ampliamente el valor obtenido de ellos.

Una **base de datos** es un conjunto de datos, que generalmente describe las actividades de una o varias organizaciones relacionadas. Por ejemplo, la base de datos de una universidad puede contener información sobre:

- *Entidades* como alumnos, profesores, asignaturas y aulas.
- *Relaciones* entre esas entidades, como la matriculación de los alumnos en las diversas asignaturas, los profesores que imparten cada asignatura y el empleo de las aulas para las diferentes asignaturas.

Un **sistema gestor de bases de datos**, o **SGBD**, es el software diseñado para colaborar en el mantenimiento y empleo de grandes conjuntos de datos. La necesidad de este tipo de sistemas, así como su uso, está aumentando rápidamente. La alternativa al empleo de un SGBD es almacenar los datos en archivos y escribir código específico para una aplicación que los gestione. El empleo de un SGBD presenta varias ventajas de importancia, como se verá en el Apartado 1.4.

1.1 GESTIÓN DE DATOS

El objetivo de este libro es presentar una introducción en profundidad a los sistemas gestores de bases de datos, con el énfasis puesto en la manera de *diseñar* bases de datos y *usar* los SGBD de manera efectiva. No cabe sorprenderse de que muchas de las decisiones sobre la manera de usar un SGBD concreto para una aplicación dada dependan de las posibilidades que ese SGBD soporte eficientemente. Por tanto, para usar bien un SGBD, es necesario comprender también la manera en que *funciona*.

Actualmente se utilizan muchos tipos de sistemas gestores de bases de datos, pero este libro se concentrará en los **sistemas gestores de bases de datos relacionales (SGBDR)**, que son, por mucho, el tipo de SGBD dominante hoy en día. Las preguntas siguientes se responden en los capítulos principales de este libro:

1. **Diseño de bases de datos y desarrollo de aplicaciones.** ¿Cómo puede un usuario describir una empresa real (por ejemplo, una universidad) en términos de los datos alma-

cenados en un SGBD? ¿Qué factores se deben tomar en consideración a la hora de decidir la manera de organizar los datos almacenados? ¿Cómo se pueden desarrollar aplicaciones basadas en SGBD? (Capítulos 2, 3, 6, 7, 12, 13 y 14).

2. **Análisis de datos.** ¿Cómo puede un usuario responder a preguntas sobre la empresa planteando consultas sobre los datos del SGBD? (Capítulos 4 y 5)¹.
3. **Concurrencia.** ¿Cómo permite un SGBD que muchos usuarios tengan acceso concurrente a los datos? (Capítulo 8).
4. **Eficiencia y escalabilidad.** ¿Cómo almacena un SGBD conjuntos de datos de gran tamaño y responde preguntas sobre esos datos de manera eficiente? (Capítulos 9, 10 y 11).

Los capítulos posteriores tratan temas importantes y que se hallan en rápida evolución, como los almacenes de datos (*data warehouses*) y las consultas complejas de ayuda a la toma de decisiones, la minería de datos (*data mining*), las bases de datos y la recuperación de la información, los repositorios XML y las bases de datos orientadas a objetos.

En el resto de este capítulo se introducirán varios temas. En el Apartado 1.2 se comenzará una breve historia de este campo científico y una discusión sobre el papel de la gestión de las bases de datos en los sistemas de información modernos. Posteriormente se identificarán las ventajas del almacenamiento de datos en SGBD en lugar de en sistemas de archivos en el Apartado 1.3, y se discutirán las ventajas del empleo de SGBD para gestionar datos en el Apartado 1.4. En el Apartado 1.5 se considerará la manera en que se debe organizar y almacenar en un SGBD la información relativa a las empresas. Probablemente los usuarios piensen en esta información en términos de alto nivel que se correspondan con las entidades de la organización y con sus relaciones, mientras que los SGBD acaban almacenando los datos en forma de (muchísimos) bits. La distancia entre la manera en que los usuarios piensan en sus datos y el modo en que se acaban almacenando se salva mediante varios *niveles de abstracción* soportados por los SGBD. De manera intuitiva, los usuarios pueden empezar describiendo los datos en términos de un nivel bastante elevado y refinar luego esa descripción en lo que haga falta cuando se consideren detalles tanto de su almacenamiento como de su representación.

En el Apartado 1.6 se considera la manera en que los usuarios pueden recuperar los datos almacenados en los SGBD y la necesidad de técnicas que calculen de manera eficiente las respuestas a las preguntas que afecten a esos datos. En el Apartado 1.7 se ofrece una visión general del modo en que los SGBD permiten que múltiples usuarios accedan concurrentemente a los datos.

Luego se describe brevemente la arquitectura de los SGBD en el Apartado 1.8 y se mencionan diversos grupos de personas asociadas con el desarrollo y empleo de los SGBD en el Apartado 1.9.

1.2 PERSPECTIVA HISTÓRICA

Desde los primeros días de la informática el almacenamiento y el tratamiento de los datos han sido el centro de atención de importantes aplicaciones. El primer SGBD de finalidad

¹En Internet también se dispone de un capítulo en inglés sobre QBE (Query-by-Example).

6 Sistemas de gestión de bases de datos

generalista, diseñado por Charles Bachman en General Electric a principios de los años sesenta del siglo veinte se denominó Almacén integrado de datos (Integrated Data Store). Aportó la base para el *modelo de datos en red*, que se normalizó en la Conferencia sobre lenguajes de sistemas de datos (Conference on Data Systems Languages, CODASYL) y tuvo gran influencia en los sistemas de bases de datos a lo largo de los años sesenta del siglo veinte. Bachman fue el primer ganador del Premio Turing de la ACM (el equivalente informático al Premio Nobel) por su trabajo en el campo de las bases de datos; lo recibió en 1973.

A finales de los años sesenta del siglo veinte IBM desarrolló el SGBD Sistema de Gestión de la Información (Information Management System, IMS), empleado incluso hoy en día en muchas instalaciones importantes. IMS constituyó la base para un entramado alternativo para la representación de los datos denominado *modelo jerárquico de datos*. El sistema SABRE para la realización de reservas en las líneas aéreas fue desarrollado conjuntamente por American Airlines e IBM por aquella época, y permitía que varias personas tuvieran acceso a los mismos datos mediante una red informática. Resulta interesante que hoy en día se utilice el mismo sistema SABRE para el funcionamiento de servicios de viajes basados en Internet tan conocidos como Travelocity.

En 1970, Edgar Codd, del Laboratorio de Investigación de San José de IBM, propuso un nuevo entramado de representación de los datos denominado *modelo relacional de datos*. Este modelo ha demostrado ser un hito en el desarrollo de los sistemas de bases de datos: provocó el rápido desarrollo de varios SGBD basados en el modelo relacional, junto con una amplia variedad de resultados teóricos que han dado sólidos fundamentos a esta rama de la ciencia. Codd ganó el Premio Turing de 1981 por su trabajo pionero. Los sistemas de bases de datos maduraron como disciplina académica, y la popularidad de los SGBD cambió el paisaje comercial. Sus ventajas fueron ampliamente reconocidas y el empleo de SGBD para la gestión de los datos empresariales se volvió algo habitual.

En los años ochenta del siglo veinte el modelo relacional consolidó su posición como paradigma dominante de los SGBD y los sistemas de bases de datos siguieron ampliando su ámbito de aplicación. El lenguaje de consultas SQL para las bases de datos relacionales, desarrollado como parte del proyecto System R de IBM, es en la actualidad el lenguaje de consultas estándar. SQL fue normalizado a finales de los años ochenta del siglo veinte y la norma actual, SQL:1999, ha sido adoptada por el Instituto Nacional Americano de Normalización (American National Standards Institute, ANSI) y la Organización Internacional para la Normalización (International Organization for Standardization, ISO). Puede afirmarse que la forma más utilizada de programación concurrente es la ejecución concurrente de programas de bases de datos (denominados *transacciones*). Los usuarios escriben los programas como si se fueran a ejecutar de manera independiente, y la responsabilidad de ejecutarlos de manera concurrente recae en el SGBD. James Gray ganó el Premio Turing 1999 por sus aportaciones a la gestión de las transacciones de las bases de datos.

A finales de los años ochenta y en los años noventa del siglo veinte se realizaron avances en muchos aspectos de los sistemas de bases de datos. Se llevó a cabo gran cantidad de investigaciones en lenguajes de consulta más potentes y en modelos de datos más ricos, con el énfasis puesto en el soporte de complejos análisis de todas las partes de una empresa. Varios fabricantes (por ejemplo, DB2 de IBM, Oracle 8 y UDS de Informix²) ampliaron sus

²Informix fue adquirida recientemente por IBM.

sistemas con la posibilidad de almacenar nuevos tipos de datos como las imágenes y el texto, y la de formular consultas más complejas. Muchos fabricantes han desarrollado sistemas especializados para la creación de *almacenes de datos*, la consolidación de datos procedentes de varias bases de datos y la ejecución de análisis especializados.

Un fenómeno interesante es la aparición de varios paquetes de **planificación de recursos empresariales** (enterprise resource planning, ERP) y de **planificación de recursos de gestión** (management resource planning, MRP), que añaden una capa sustancial de características orientadas a las aplicaciones por encima de los SGBD. Entre los paquetes más usados hay sistemas de Baan, Oracle, PeopleSoft, SAP y Siebel. Estos paquetes identifican un conjunto de tareas frecuentes (por ejemplo, la gestión de inventarios, la planificación de los recursos humanos, el análisis financiero) que se llevan a cabo en gran número de organizaciones y ofrecen una capa general de aplicaciones para desempeñarlas. Los datos se almacenan en un SGBD relacional y la capa de aplicaciones se puede personalizar para diferentes empresas, lo que permite reducir los costes globales para las empresas en comparación con el coste de creación de la capa de aplicaciones desde el principio.

Lo que quizás resulte más significativo es que los SGBD han entrado en la Era de Internet. Mientras la primera generación de sitios Web guardaba sus datos exclusivamente en archivos del sistema operativo, el empleo de SGBD para almacenar los datos a que se tiene acceso mediante los navegadores Web se está generalizando. Las consultas se generan mediante formularios accesibles por Web y se da formato a las respuestas mediante un lenguaje de marcas como HTML para que se muestren con facilidad en los navegadores. Todos los fabricantes de bases de datos están añadiendo a sus SGBD características dirigidas a hacerlos más adecuados para su implantación en Internet.

La gestión de bases de datos sigue adquiriendo importancia a medida que se van poniendo en línea más datos y se hacen cada vez más accesibles mediante las redes de ordenadores. Hoy en día este campo está siendo impulsado por visiones excitantes como las bases de datos multimedia, el vídeo interactivo, las corrientes de datos, las bibliotecas digitales, buen número de proyectos científicos como el esfuerzo de elaboración del mapa del genoma humano y el proyecto del Sistema de observación de la Tierra de la NASA (Earth Observation System) y el deseo de las empresas de consolidar sus procesos de toma de decisiones y *explorar* sus repositorios de datos en búsqueda de información útil sobre sus negocios. Comercialmente, los sistemas gestores de bases de datos representan uno de los mayores y más vigorosos segmentos del mercado. Por tanto, el estudio de los sistemas de bases de datos puede resultar muy provechoso por muchos motivos.

1.3 SISTEMAS DE ARCHIVOS Y SGBD

Para comprender la necesidad de los SGBD, considérese la siguiente situación: una empresa tiene un gran conjunto de datos (digamos que unos 500 GB³) sobre empleados, departamentos, productos, ventas, etcétera. A estos datos tienen acceso de manera concurrente varios empleados. Las preguntas sobre los datos se deben responder rápidamente, las modificaciones

³Un kilobyte (KB) equivale a 1024 bytes, un megabyte (MB) a 1024 KB, un gigabyte (GB) a 1024 MB, un terabyte (TB) a 1024 GB y un petabyte (PB) a 1024 terabytes.

8 Sistemas de gestión de bases de datos

de los datos realizadas por los diferentes usuarios deben aplicarse de manera consistente y el acceso a ciertas partes de los datos (por ejemplo, los sueldos) debe estar restringido.

Se puede intentar gestionar los datos almacenándolos en archivos del sistema operativo. Este enfoque tiene muchos inconvenientes y, entre ellos, los siguientes:

- Probablemente no se disponga de 500 GB de memoria principal para albergar todos los datos. Por tanto, se deben guardar los datos en un dispositivo de almacenamiento como discos o cintas y llevar las partes de importancia a la memoria principal para su procesamiento cuando resulte necesario.
- Aunque se dispusiera de 500 GB de memoria principal, en los sistemas informáticos con direccionamiento de treinta y dos bits no se puede hacer referencia directamente más que a unos 4 GB de datos. Hay que programar algún método para identificar todos los datos.
- Hay que escribir programas especiales para responder a cada pregunta que puedan formular los usuarios sobre esos datos. Es muy probable que esos programas sean complejos debido al gran volumen de datos que hay que analizar.
- Hay que proteger los datos de las modificaciones inconsistentes realizadas por usuarios diferentes que acceden a los datos de manera concurrente. Si las aplicaciones deben afrontar los detalles de ese acceso concurrente se incrementa notablemente su complejidad.
- Hay que garantizar que los datos vuelvan a un estado consistente si el sistema falla mientras se está realizando alguna modificación.
- Los sistemas operativos sólo ofrecen para la seguridad un mecanismo de contraseñas. Esto no es lo bastante flexible para la aplicación de directivas de seguridad en las que los diferentes usuarios tengan permiso para el acceso a diferentes subconjuntos de los datos.

Los SGBD son aplicaciones software diseñadas para facilitar las tareas mencionadas. Al almacenar los datos en un SGBD en vez de en un conjunto de archivos del sistema operativo, se pueden utilizar las características del SGBD para gestionar los datos de un modo robusto y eficiente. A medida que crecen el volumen de los datos y el número de usuarios —en las bases de datos corporativas actuales son habituales los centenares de gigabytes y los millares de usuarios— el apoyo de los SGBD se vuelve indispensable.

1.4 VENTAJAS DE LOS SGBD

El empleo de SGBD para gestionar los datos tiene muchas ventajas:

- **Independencia con respecto a los datos.** Los programas de las aplicaciones no deben, en principio, exponerse a los detalles de la representación y el almacenamiento de los datos. El SGBD ofrece una vista abstracta de los datos que oculta esos detalles.
- **Acceso eficiente a los datos.** Los SGBD emplean gran variedad de técnicas sofisticadas para almacenar y recuperar los datos de manera eficiente. Esta característica resulta especialmente importante si los datos se guardan en dispositivos de almacenamiento externos.

- **Integridad y seguridad de los datos.** Si siempre se tiene acceso a los datos mediante el SGBD, éste puede hacer que se cumplan las restricciones de integridad. Por ejemplo, antes de introducir la información salarial de un empleado, el SGBD puede comprobar que no se haya superado el presupuesto del departamento. Además, puede hacer que se cumplan los *controles de acceso* que determinan los datos que son visibles para las diferentes clases de usuarios.
- **Administración de los datos.** Cuando varios usuarios comparten los mismos datos, la centralización de la administración de esos datos puede ofrecer mejoras significativas. Los profesionales experimentados que comprenden la naturaleza de los datos que se gestionan y la manera en que los utilizan los diferentes grupos de usuarios, pueden responsabilizarse de la organización de la representación de los datos para minimizar la redundancia y mejorar el almacenamiento de los datos para hacer que la recuperación sea eficiente.
- **Acceso concurrente y recuperación en caso de fallo.** Los SGBD programan los accesos concurrentes a los datos de tal manera que los usuarios puedan creer que sólo tiene acceso a los datos un usuario a la vez. Además, los SGBD protegen a los usuarios de los efectos de los fallos del sistema.
- **Reducción del tiempo de desarrollo de las aplicaciones.** Evidentemente, los SGBD soportan funciones importantes que son comunes con muchas aplicaciones que tienen acceso a los datos del SGBD. Esto, junto con la interfaz de alto nivel con los datos, facilita el rápido desarrollo de aplicaciones. También es probable que las aplicaciones de los SGBD sean más robustas que las aplicaciones independientes similares, ya que el SGBD maneja muchas tareas importantes (y no hay que depurarlas y probarlas en la aplicación).

Dadas todas estas ventajas, ¿puede haber alguna razón para *no* utilizar un SGBD? A veces, sí. Los SGBD son piezas de software complejas, optimizados para ciertos tipos de cargas de trabajo (por ejemplo, la respuesta a consultas complejas o el manejo de muchas solicitudes concurrentes), y puede que su rendimiento no sea el adecuado para ciertas aplicaciones especializadas. Entre los ejemplos están las aplicaciones con estrictas restricciones de tiempo real o tan sólo unas pocas operaciones críticas bien definidas para las cuales hay que escribir un código a medida eficiente. Otro motivo para no emplear un SGBD es que puede que una aplicación necesite manipular los datos de maneras no soportadas por el lenguaje de consultas. En esas situaciones, la vista abstracta de los datos ofrecida por el SGBD no coincide con las necesidades de la aplicación y, en realidad, estorba. A modo de ejemplo, las bases de datos relacionales no soportan el análisis flexible de datos de texto (aunque los fabricantes estén ampliando actualmente sus productos en esa dirección). Si el rendimiento especializado o las exigencias del tratamiento de los datos son fundamentales para una aplicación, ésta puede no utilizar un SGBD, especialmente si las ventajas añadidas de los SGBD (por ejemplo, la flexibilidad de las consultas, la seguridad, el acceso concurrente y la recuperación de fallos) no son necesarias. En la mayor parte de las situaciones que requieren la gestión de datos a gran escala, no obstante, los SGBD se han convertido en una herramienta indispensable.

1.5 DESCRIPCIÓN Y ALMACENAMIENTO DE DATOS EN LOS SGBD

El usuario de un SGBD está, en última instancia, preocupado por alguna empresa real, y los datos que hay que guardar describen diversos aspectos de esa empresa. Por ejemplo, en las universidades hay alumnos, profesores y asignaturas, y los datos de las bases de datos de las universidades describen esas entidades y sus relaciones.

Un **modelo de datos** es un conjunto de estructuras descriptivas de datos de alto nivel que oculta muchos detalles de almacenamiento de bajo nivel. Los SGBD permiten a los usuarios definir los datos que se van a almacenar en términos de un modelo de datos. La mayor parte de los sistemas actuales de gestión de bases de datos se basan en el **modelo relacional de datos**, en el que se centrará este libro.

Aunque el modelo de datos del SGBD oculta muchos detalles, pese a todo está más cercano al modo en que el SGBD almacena los datos que al modo en que el usuario piensa en la empresa subyacente. Los **modelos semánticos de datos** son modelos de datos de alto nivel más abstractos que facilitan que los usuarios obtengan una buena descripción inicial de los datos de las empresas. Estos modelos contienen una amplia variedad de estructuras que ayudan a describir la situación real de las aplicaciones. No se pretende que los SGBD soporten directamente todas esas estructuras; suele crearse alrededor de un modelo de datos que tiene tan sólo unas cuantas estructuras básicas, como puede ser el modelo relacional. El diseño de bases de datos en términos de un modelo semántico sirve como útil punto de partida y se traduce posteriormente en el diseño de una base de datos en términos del modelo de datos que soporta realmente el SGBD.

Un modelo semántico de datos muy utilizado, denominado modelo entidad-relación (ER) nos permitirá denotar de manera gráfica las entidades y las relaciones existentes entre ellas. El modelo ER se trata en el Capítulo 2.

1.5.1 El modelo relacional

En este apartado se ofrece una breve introducción al modelo relacional. La estructura central para la descripción de los datos en este modelo son las **relaciones**, que se pueden considerar conjuntos de **registros**.

La descripción de los datos en términos de un modelo de datos se denomina **esquema**. En el modelo relacional los esquemas de las relaciones especifican su nombre, el nombre de cada **campo** (o **atributo** o **columna**) y el tipo de cada campo. A modo de ejemplo, puede que la información sobre los alumnos de la base de datos de una universidad se guarde en una relación con el esquema siguiente:

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

El esquema anterior indica que cada registro de la relación Alumnos tiene cinco campos, con los nombres y tipos de los campos que se indican. En la Figura 1.1 puede verse un ejemplar de la relación Alumnos.

Un ejemplo de mal diseño. El esquema relacional Alumnos muestra una mala decisión de diseño; *nunca* se deben crear campos como *edad*, cuyo valor cambia constantemente. Una opción mejor hubiera sido *FDN* (por *fecha de nacimiento*); la edad se puede calcular a partir de ella. No obstante, en los ejemplos se seguirá empleando *edad*, ya que los hace más fáciles de leer.

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

Figura 1.1 Ejemplar de la relación Alumnos

Cada fila de la relación Alumnos es un registro que describe a un alumno. La descripción no está completa —por ejemplo, no se ha incluido la altura de cada alumno— pero, presumiblemente, resulta adecuada para las aplicaciones deseadas de una base de datos universitaria. Cada fila sigue el esquema de la relación Alumnos. El esquema puede, por tanto, considerarse como una plantilla para la descripción de los alumnos.

Se puede hacer más precisa la descripción de un conjunto de alumnos especificando **restricciones de integridad**, que son condiciones que deben satisfacer los registros de una relación. Por ejemplo, se podría especificar que cada alumno tenga un valor único de *ide*. Obsérvese que no se puede capturar esta información añadiendo simplemente otro campo al esquema Alumnos. Por tanto, la posibilidad de especificar la unicidad de los valores de un campo aumenta la precisión con que se pueden describir los datos. La expresividad de las estructuras disponibles para la especificación de restricciones de integridad es un aspecto importante de los modelos de datos.

Otros modelos de datos

Además del modelo de datos relacional (que se utiliza en numerosos sistemas, como DB2 de IBM, Informix, Oracle, Sybase, Access de Microsoft, FoxBase, Paradox, Tandem y Teradata) hay otros modelos de datos importantes, como el modelo jerárquico (que se emplea, por ejemplo en IDS y en IDMS), el modelo orientado a objetos (que se usa, por ejemplo, en Objectstore y en Versant) y el modelo relacional orientado a objetos (que se utiliza, por ejemplo, en productos de SGBD de IBM, Informix, ObjectStore, Oracle, Versant y otros). Aunque muchas bases de datos emplean los modelos jerárquico y de red, y los sistemas basados en los modelos orientado a objetos y relacional orientado a objetos están ganando aceptación en el mercado, el modelo dominante en la actualidad es el relacional.

En este libro nos centraremos en el modelo relacional debido a su empleo generalizado y a su importancia. En realidad, el modelo relacional orientado a objetos, que está ganando

12 Sistemas de gestión de bases de datos

popularidad, es un esfuerzo para combinar las mejores características de los modelos relacional y orientado a objetos, y es necesaria una buena comprensión del modelo relacional para entender los conceptos relacionales orientados a objetos.

1.5.2 Niveles de abstracción en los SGBD

Los datos en los SGBD se describen en tres niveles de abstracción, como puede verse en la Figura 1.2. La descripción de las bases de datos consta de un esquema en cada uno de esos tres niveles de abstracción: *conceptual*, *físico* y *externo*.

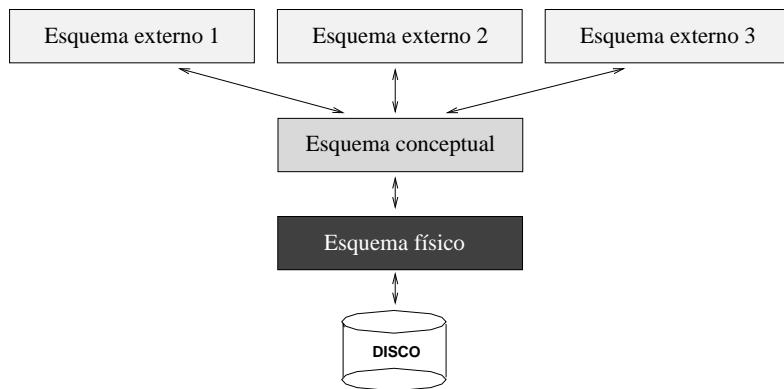


Figura 1.2 Niveles de abstracción en los SGBD

Los **lenguajes de definición de datos** (LDD) se emplean para definir los esquemas externo y conceptual. Las posibilidades como LDD del lenguaje de bases de datos más utilizado, SQL, se tratan en el Capítulo 3. Todos los fabricantes de SGBD soportan también las órdenes de SQL para la descripción de aspectos del esquema físico, pero esas órdenes no forman parte de la norma del lenguaje SQL. La información sobre los esquemas conceptual, externo y físico se guarda en los **catálogos del sistema**. Los tres niveles de abstracción se tratarán en el resto de este apartado.

Esquema conceptual

El **esquema conceptual** (denominado a veces **esquema lógico**) describe los datos almacenados en términos del modelo de datos del SGBD. En un SGBD relacional, el esquema conceptual describe todas las relaciones almacenadas en la base de datos. En la base de datos universitaria del ejemplo, esas relaciones contienen información sobre *entidades*, como los alumnos y el profesorado, y sobre *relaciones*, como la matrícula de los alumnos en las asignaturas. Todas las entidades alumno se pueden describir empleando registros de la relación Alumnos, como ya se ha visto. De hecho, cada conjunto de entidades y cada conjunto de relaciones pueden describirse como relaciones, lo que lleva al siguiente esquema conceptual:

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

```

Profesores(idp: string, nombrep: string, sueldo: real)
Asignaturas(ida: string, nombrea: string, créditos: integer)
Aulas(nau: integer, dirección: string, capacidad: integer)
Matriculado(ide: string, ida: string, curso: string)
Imparte(idp: string, ida: string)
Impartida_en(ida: string, nau: integer, hora: string)

```

La elección de las relaciones, y la de los campos de cada relación, no resulta siempre evidente, y el proceso de búsqueda de un buen esquema conceptual se denomina **diseño conceptual de bases de datos**. El diseño conceptual de bases de datos se trata en los Capítulos 2 y 12.

Esquema físico

El **esquema físico** especifica detalles adicionales del almacenamiento. Esencialmente, el esquema físico resume el modo en que las relaciones descritas en el esquema conceptual se guardan realmente en dispositivos de almacenamiento secundario como discos y cintas.

Es necesario decidir la organización de archivos que se va a utilizar para guardar las relaciones y crear estructuras de datos auxiliares, denominadas **índices**, para acelerar las operaciones de recuperación. Un ejemplo de esquema físico para la base de datos universitaria es el siguiente:

- Almacenar todas las relaciones como archivos de registros sin ordenar. (Cada archivo de un SGBD es un conjunto de registros o un conjunto de páginas, en vez de una cadena de caracteres como en los sistemas operativos.)
- Crear índices en la primera columna de las relaciones Alumnos, Profesores y Asignaturas, en la columna *sueldo* de Profesores y en la columna *capacidad* de Aulas.

Las decisiones sobre el esquema físico se basan en conocer el modo en que se suele tener acceso a los datos. El proceso de obtención de un buen esquema físico se denomina **diseño físico de bases de datos**. El diseño físico de bases de datos se trata en el Capítulo 13.

Esquema externo

Los **esquemas externos**, que suelen serlo también en términos del modelo de datos del SGBD, permiten personalizar (y autorizar) el acceso a los datos a los usuarios y grupos de ellos. Cualquier base de datos tiene exactamente un esquema conceptual y un esquema físico, porque sólo tiene guardado un conjunto de relaciones, pero puede tener varios esquemas externos, cada uno de ellos adaptado a un grupo de usuarios concreto. Cada esquema externo consiste en un conjunto de una o varias **vistas** y relaciones del esquema conceptual. Una vista es, conceptualmente, una relación pero los registros de la vista no se guardan en el SGBD. Por el contrario, se calculan empleando una definición de la vista en términos de las relaciones guardadas en el SGBD. Las vistas se tratan con más detalle en los Capítulos 3 y 16.

14 Sistemas de gestión de bases de datos

El diseño del esquema externo se guía por las necesidades del usuario final. Por ejemplo, puede que se desee que los alumnos averigüen el nombre de los profesores que imparten cada curso y la matrícula de cada curso. Esto puede hacerse definiendo la vista siguiente:

```
Infoasignatura(ida: string, nombrep: string, matrícula: integer)
```

Los usuarios pueden tratar la vista igual que a una relación y formularle preguntas sobre sus registros. Aunque estos registros no se guarden de manera explícita, se calculan a medida que hacen falta. Infoasignatura no se incluyó en el esquema conceptual porque se puede calcular a partir de las relaciones de ese esquema, y guardarla también resultaría redundante. Esa redundancia, además del espacio desaprovechado, podría dar lugar a inconsistencias. Por ejemplo, se puede insertar una tupla en la relación Matriculado que indique que un alumno dado se ha matriculado en un curso concreto, sin incrementar el valor del campo *matrícula* del registro correspondiente de Infoasignatura (si esta última forma también parte del esquema conceptual y sus tuplas se guardan en el SGBD).

1.5.3 Independencia con respecto a los datos

Una ventaja muy importante del empleo de un SGBD es que ofrece **independencia con respecto a los datos**. Es decir, los programas de aplicación quedan aislados de las modificaciones debido al modo en que se estructuran y se guardan los datos. La independencia con respecto a los datos se consigue mediante el empleo de los tres niveles de abstracción de los datos; en concreto, el esquema conceptual y el externo ofrecen claras ventajas en este campo.

Las relaciones en el esquema externo (relaciones de vistas) se generan, en principio, a petición de los usuarios a partir de las relaciones correspondientes del esquema conceptual⁴. Si se reorganiza la capa subyacente, es decir, se modifica el esquema conceptual, se puede modificar la definición de la relación de vistas para que esa relación se siga calculando como antes. Por ejemplo, supóngase que la relación Profesores de la base de datos universitaria se sustituye por las dos relaciones siguientes:

```
Profesores_públicos(idp: string, nombrep: string, despacho: integer)
```

```
Profesores_privados(idp: string, sueldo: real)
```

De manera intuitiva, cierta información confidencial sobre el profesorado se ha colocado en una relación diferente y se ha añadido la información relativa a los despachos. Se puede redefinir la relación de vistas Infoasignatura en términos de Profesores_públicos y Profesores_privados, que contienen entre las dos toda la información de Profesores, por lo que los usuarios que consulten Infoasignatura obtendrán las mismas respuestas que antes.

Por tanto, se puede proteger a los usuarios de las modificaciones en la estructura lógica de los datos, o en la elección de las relaciones que se guardan. Esta propiedad se denomina **independencia con respecto a los datos**.

A su vez, el esquema conceptual aisla a los usuarios respecto de las modificaciones en los detalles del almacenamiento físico. Esta propiedad se conoce como **independencia física con respecto a los datos**. El esquema conceptual oculta detalles como el modo en que se

⁴En la práctica se podrían calcular con antelación y guardarse para acelerar las consultas relativas a las relaciones de vistas, pero las relaciones de vistas calculadas deben actualizarse siempre que se actualicen las relaciones subyacentes.

disponen realmente los datos en el disco, la estructura de los archivos y la elección de los índices. En tanto en cuanto el esquema conceptual siga siendo el mismo, se pueden modificar esos detalles del almacenamiento sin alterar las aplicaciones. (Por supuesto, puede que el rendimiento se vea afectado por esas modificaciones.)

1.6 LAS CONSULTAS EN LOS SGBD

La facilidad con la que se puede obtener información de las bases de datos suele determinar su valor para los usuarios. A diferencia de los sistemas antiguos de bases de datos, los sistemas relacionales de bases de datos permiten formular con facilidad una amplia gama de preguntas; esta característica ha contribuido mucho a su popularidad. Considérese la base de datos universitaria del ejemplo del Apartado 1.5.2. He aquí algunas de las preguntas que los usuarios podrían plantear:

1. ¿Cómo se llama el alumno con ID 123456?
2. ¿Cuál es el sueldo medio de los profesores que imparten la asignatura CS564?
3. ¿Cuántos alumnos se han matriculado en CS564?
4. ¿Qué proporción de los alumnos de CS564 obtuvieron una nota superior a notable?
5. ¿Hay algún alumno con nota menor de 6,0 matriculado en CS564?

Esas preguntas relativas a los datos guardados en el SGBD se denominan **consultas**. Los SGBD proporcionan un lenguaje especializado, denominado **lenguaje de consultas**, en el que se pueden formular las consultas. Una característica muy atractiva del modelo relacional es que soporta lenguajes de consulta potentes. El **cálculo relacional** es un lenguaje de consultas formal basado en la lógica matemática, y las consultas en ese lenguaje tienen un significado preciso e intuitivo. El álgebra relacional es otro lenguaje formal de consultas, basado en un conjunto de **operadores** para la manipulación de las relaciones, que es equivalente en potencia al cálculo relacional.

Los SGBD tiene mucho cuidado en evaluar las consultas de la manera más eficiente posible. Por supuesto, la eficiencia de la evaluación de las consultas viene determinada en gran parte por la manera en que los datos se han almacenado físicamente. Se pueden utilizar índices para acelerar muchas consultas —de hecho, una buena elección de índices para las relaciones subyacentes puede acelerar cada una de las consultas de la lista anterior—. El almacenamiento de datos y los índices se tratan en los Capítulos 9, 10 y 11.

Los SGBD permiten a los usuarios crear, modificar y consultar datos mediante los **lenguajes de manipulación de datos** (LMD). Por tanto, el lenguaje de consultas es tan sólo una parte del LMD, que también proporciona estructuras para añadir, eliminar y modificar datos. Las características LMD de SQL se tratan en el Capítulo 5. LMD y LDD se conocen en conjunto como **sublenguaje de datos** cuando se incluyen en un **lenguaje anfitrión** (como, por ejemplo, C o COBOL).

1.7 GESTIÓN DE TRANSACCIONES

Considérese una base de datos que guarde información sobre reservas de billetes de avión. En cualquier momento dado es posible (y probable) que varios agentes de viajes estén buscando información sobre plazas disponibles en diferentes vuelos y haciendo nuevas reservas de billetes. Cuando varios usuarios tienen acceso (y, posiblemente, modifican) una base de datos de manera concurrente, el SGBD debe ordenar sus solicitudes con sumo cuidado para evitar conflictos. Por ejemplo, cuando un agente de viajes busca el Vuelo 100 de un día dado y halla una plaza libre, puede que otro agente de viajes esté haciendo al mismo tiempo una reserva para esa misma plaza, lo que deja obsoleta la información vista por el primero.

Otro ejemplo de uso concurrente son las bases de datos de las entidades bancarias. Mientras el programa de aplicación de un usuario está calculando el total de depósitos, puede que otra aplicación transfiera dinero de una cuenta que la primera acabe de “ver” a una cuenta que todavía no ha “visto”, lo que hace que el total parezca mayor de lo que debería. Evidentemente, no se debe permitir que ocurran tales anomalías. Sin embargo, impedir el acceso concurrente puede degradar el rendimiento.

Además, los SGBD deben proteger a los usuarios de los efectos de los fallos del sistema asegurando que todos los datos (y el estado de las aplicaciones activas) vuelvan a un estado consistente cuando se reinicie el sistema tras un fallo. Por ejemplo, si un agente de viajes pide que se lleve a cabo una reserva y el SGBD responde diciendo que esa reserva se ha completado, la reserva no se debe perder si el sistema falla. Por otro lado, si el SGBD no ha respondido todavía a esa solicitud pero está llevando a cabo las necesarias modificaciones a los datos cuando se produce el fallo, esas modificaciones parciales se deben deshacer cuando el sistema vuelve a estar en funcionamiento.

Una **transacción** es *cualquier ejecución* de un programa de usuario en un SGBD. (Las ejecuciones subsecuentes del mismo programa generan varias transacciones.) Se trata de la unidad básica de modificación desde el punto de vista del SGBD: no se permiten transacciones parciales, y el efecto de un grupo de transacciones es equivalente a la ejecución en serie de todas las transacciones. A continuación se describe brevemente la manera en que se garantizan estas propiedades.

1.7.1 Ejecución concurrente de las transacciones

Una importante tarea de los SGBD es la programación de los accesos concurrentes a los datos de modo que cada usuario pueda ignorar con seguridad el hecho de que otros tienen acceso a los datos de manera concurrente. La importancia de esta tarea no puede subestimarse, ya que las bases de datos suelen estar compartidas por gran número de usuarios, que remiten sus consultas al SGBD de manera independiente y, sencillamente, no puede esperarse que afronten las modificaciones arbitrarias que realicen otros usuarios de manera concurrente. El SGBD permite que los usuarios crean que sus programas se ejecutan aisladamente, uno tras otro en el orden escogido por el SGBD. Por ejemplo, si el programa que deposita efectivo en una cuenta se remite al SGBD al mismo tiempo que otro programa que retira dinero de esa misma cuenta, el SGBD puede ejecutar cualquiera de los dos, pero sus procesos no se intercalarán de modo que interfieran entre sí.

Los **protocolos de bloqueo** son conjuntos de reglas que debe seguir cada transacción (y que el SGBD debe hacer que se cumplan) para garantizar que, aunque las acciones de varias transacciones se intercalen, el efecto neto sea idéntico a la ejecución de todas las transacciones en un orden consecutivo dado. Un **bloqueo** es un mecanismo empleado para controlar el acceso a los objetos de la base de datos. Los SGBD suelen soportar dos tipos de **bloqueos**: los **compartidos** sobre un objeto pueden establecerlos dos transacciones diferentes al mismo tiempo, pero el **bloqueo exclusivo** de un objeto garantiza que ninguna otra transacción establezca *un* bloqueo sobre ese objeto.

Supóngase que se sigue el siguiente protocolo de bloqueo: *todas las transacciones comienzan por la obtención de un bloqueo compartido sobre cada objeto de datos que deban modificar y luego liberan todos los bloqueos tras completar todas las acciones*. Considérense dos transacciones T_1 y T_2 tales que T_1 desea modificar un objeto de datos y T_2 desea leer ese mismo objeto. De manera intuitiva, si la solicitud por T_1 de un bloqueo exclusivo se concede en primer lugar, T_2 no puede seguir adelante hasta que T_1 libere ese bloqueo, ya que el SGBD no concederá la solicitud de un bloqueo compartido por parte de T_2 hasta entonces. Por tanto, todas las acciones de T_1 se completarán antes de que se inicie ninguna acción de T_2 . Se considerarán los bloqueos con más detalle en el Capítulo 8.

1.7.2 Las transacciones no completadas y los fallos del sistema

Las transacciones pueden interrumpirse antes de completarse por gran variedad de motivos como, por ejemplo, un fallo del sistema. Los SGBD deben garantizar que las modificaciones llevadas a cabo por esas transacciones no completadas se eliminen de las bases de datos. Por ejemplo, si el SGBD se halla en trance de transferir dinero de la cuenta A a la cuenta B y ha cargado la operación en la primera cuenta pero todavía no la ha anotado en la segunda cuando se produce el fallo, se debe devolver el dinero cargado a la cuenta A cuando el sistema se recupere del fallo.

Para ello, el SGBD mantiene un **registro** de todas las operaciones de escritura en la base de datos. Una propiedad fundamental de ese registro es que cada acción de escritura debe registrarse en el registro (en disco) *antes* de que la modificación correspondiente se refleje en la propia base de datos —en caso contrario, si el sistema fallase justo tras la realización de la modificación de la base de datos pero antes de que ésta se reflejara en el registro, el SGBD no sería capaz de detectarla y deshacerla—. Esta propiedad se denomina **registro de escritura previa** (Write-Ahead Log, **WAL**). Para garantizarla, el SGBD debe poder obligar de manera selectiva a que se guarde en el disco una página que se halle en la memoria.

El registro se utiliza también para garantizar que las modificaciones llevadas a cabo por las transacciones completadas con éxito no se pierdan debido a fallos del sistema. La devolución de la base de datos a un estado consistente tras un fallo del sistema puede ser un proceso lento, ya que el SGBD debe garantizar que el efecto de todas las transacciones que se completaron antes del fallo se restaure, y que el de las transacciones no completadas se deshaga. El tiempo necesario para recuperarse de un fallo puede reducirse obligando de manera periódica a que cierta información se guarde en disco; esas operaciones periódicas se denominan **puntos de revisión**.

1.7.3 Puntos a destacar

En resumen, hay tres puntos que recordar en relación con el soporte del control de la concurrencia y las recuperaciones de los SGBD:

1. Todos los objetos que lee o escribe una transacción se bloquean antes de modo compartido o exclusivo, respectivamente. El establecimiento de un bloqueo sobre un objeto restringe su disponibilidad para otras transacciones y, por tanto, afecta al rendimiento.
2. Para un mantenimiento eficiente del registro, el SGBD debe poder obligar de manera selectiva a que un conjunto de páginas de la memoria principal se guarde en el disco. El soporte de esta operación por parte del sistema operativo no siempre resulta satisfactorio.
3. El establecimiento de puntos de revisión periódicos puede reducir el tiempo necesario para la recuperación de un fallo. Por supuesto, esto debe contraponerse al hecho de que el establecimiento de puntos de revisión demasiado próximos hace más lenta la ejecución normal.

1.8 ARQUITECTURA DE LOS SGBD

La Figura 1.3 muestra la arquitectura (con algunas simplificaciones) de un SGBD típico basado en el modelo relacional de datos.

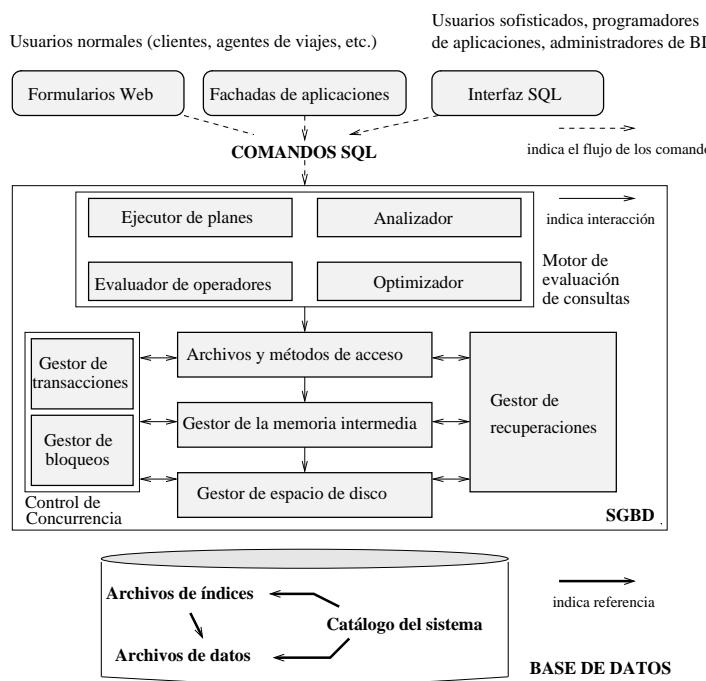


Figura 1.3 Arquitectura de un SGBD

El SGBD acepta las órdenes SQL generadas por gran variedad de interfaces de usuario, produce planes de evaluación de consultas, ejecuta esos planes contra la base de datos y devuelve las respuestas. (Esto no es más que una simplificación: las órdenes de SQL pueden estar incluidas en programas de aplicaciones de lenguajes anfitriones como, por ejemplo, programas de Java o de COBOL. Estos aspectos se pasan por alto para concentrarnos en la funcionalidad principal del SGBD.)

Cuando un usuario formula una consulta, se analiza y envía esta consulta a un **optimizador de consultas**, que utiliza información sobre el modo en que se guardan los datos para producir un plan de ejecución eficiente para la evaluación de esa consulta. Un **plan de ejecución** es un plan detallado para la evaluación de la consulta, representado habitualmente como un árbol de operadores relationales (con anotaciones que contienen información detallada adicional sobre los métodos de acceso que se deben emplear, etcétera). Los operadores relationales son como los elementos constitutivos de la evaluación de las consultas planteadas a los datos.

El código que implementa los operadores relationales se sitúa por encima de la capa de los archivos y los métodos de acceso. Esta capa soporta el concepto de **archivo**, que es un conjunto de páginas o de registros en los SGBD. Se admiten tanto los **archivos en montículos**, o archivos de páginas sin ordenar, así como los índices. Además de realizar el seguimiento de las páginas de los archivos, esta capa organiza la información en el interior de cada página. Las organizaciones de los archivos y de los índices se consideran en el Capítulo 9.

El código de la capa de archivos y métodos de acceso se sitúan por encima del **gestor de la memoria intermedia**, que lleva las páginas desde el disco a la memoria principal según va haciendo falta, en respuesta a las solicitudes de lectura.

La capa inferior del software de los SGBD se ocupa de la administración del espacio de disco, donde se almacenan los datos. Las capas superiores asignan, desasignan, leen y escriben las páginas (mediante las oportunas rutinas) a través de esta capa, denominada **gestor del espacio de disco**.

Los SGBD soportan la concurrencia y la recuperación de fallos mediante la cuidadosa programación de las solicitudes de los usuarios y el mantenimiento de un registro de todas las modificaciones de la base de datos. Entre los componentes del SGBD asociados al control de la concurrencia y la recuperación están el **gestor de transacciones**, que garantiza que las transacciones soliciten y liberen los bloqueos de acuerdo con el correspondiente protocolo de bloqueo y programa la ejecución de las transacciones; el **gestor de bloqueos**, que realiza un seguimiento de las solicitudes de bloqueo y concede los bloqueos sobre los objetos de la base de datos cuando quedan disponibles; y el **gestor de recuperaciones**, que es responsable del mantenimiento de un registro y de la restauración del sistema a un estado consistente tras los fallos. El gestor del espacio de disco, el gestor de la memoria intermedia, y las capas de archivos y métodos de acceso deben interactuar con estos componentes. El control de la concurrencia y la recuperación tras los fallos se estudian con detalle en el Capítulo 8.

1.9 USUARIOS DE LAS BASES DE DATOS

Existe una gran variedad de personas asociadas con la creación y el empleo de bases de datos. Evidentemente, hay **implementadores de bases de datos**, que crean software de SGBD, y **usuarios finales**, que desean guardar y utilizar datos de los SGBD. Los implementadores de bases de datos trabajan para fabricantes como IBM u Oracle. Los usuarios finales vienen de un número de campos diverso y creciente. A medida que los datos aumentan en complejidad y en volumen, y se van reconociendo como un activo fundamental, la importancia de mantenerlos en los SGBD se acepta cada vez más. Muchos usuarios finales se limitan a utilizar aplicaciones escritas por programadores de aplicaciones de bases de datos (véase más abajo) y, por tanto, no necesitan tener muchos conocimientos técnicos sobre el software de SGBD. Por supuesto, los usuarios sofisticados que hacen un uso más amplio de los SGBD, como la escritura de sus propias consultas, necesitan una comprensión más profunda de sus características.

Además de los usuarios finales y de los implementadores, hay otras dos clases de usuarios asociados con los SGBD: los *programadores de aplicaciones* y los *administradores de bases de datos*.

Los **programadores de aplicaciones de bases de datos** desarrollan paquetes que facilitan el acceso a los datos por parte de los usuarios finales, que generalmente no son profesionales de la informática, mediante lenguajes anfitriones o de datos y herramientas software que proporcionan los fabricantes de SGBD. (Entre esas herramientas figuran las herramientas para la escritura de informes, las hojas de cálculo, los paquetes de estadística y similares.) Los programas de aplicación deberían tener acceso a los datos, en teoría, mediante el esquema externo. Es posible escribir aplicaciones que tengan acceso a los datos en un nivel inferior, pero esas aplicaciones pondrían en peligro la independencia con respecto a los datos.

Las bases de datos suelen estar mantenidas por la persona que las posee y las utiliza. Sin embargo, las bases de datos corporativas suelen ser lo bastante importantes y complejas como para que la tarea de diseñarlas y mantenerlas se confíe a profesionales, denominados **administradores de bases de datos** (database administrator, **DBA**). Los DBA son responsables de muchas tareas críticas:

- **El diseño de los esquemas conceptual y físico.** El DBA es responsable de interactuar con los usuarios del sistema para comprender los datos que se van a guardar en el SGBD y la manera en que es más probable que se utilicen. Con ese conocimiento, el DBA debe diseñar el esquema conceptual (decidir las relaciones que se van a guardar) y el físico (decidir la manera de guardarlas). Puede que el DBA también diseñe las partes más utilizadas del esquema externo, aunque los usuarios hagan crecer este esquema mediante la creación de vistas adicionales.
- **Seguridad y autorización.** El DBA es responsable de garantizar que no se permita el acceso sin autorización a los datos. En general, no todo el mundo debe poder tener acceso a todos los datos. En los SGBD relacionales se puede conceder permiso a los usuarios para el acceso tan sólo a determinadas vistas y relaciones. Por ejemplo, aunque puede que se permita que los alumnos averigüen las matriculas de cada asignatura y quién las imparte, no es probable que se desee que vean los sueldos de los profesores o la información sobre las notas de los demás alumnos. El DBA puede aplicar esta política mediante la concesión a los alumnos de permiso para leer tan sólo la vista Infoasignatura.

- **Disponibilidad de los datos y recuperación en caso de fallo.** El DBA debe tomar medidas para garantizar que, en caso de fallo del sistema, los usuarios puedan seguir teniendo acceso a tanta cantidad de datos que no se hayan deteriorado como sea posible. El DBA también debe trabajar para devolver los datos a un estado consistente. Los SGBD ofrecen soporte de software para estas funciones, pero el DBA es responsable de la implementación de procedimientos para la realización periódica de copias de seguridad y para el mantenimiento de los registros de la actividad del sistema (para facilitar la recuperación en caso de fallo).
- **Ajuste de las bases de datos.** Es probable que las necesidades de los usuarios evolucionen con el tiempo. El DBA es responsable de modificar la base de datos, en especial los esquemas conceptual y físico, para garantizar un rendimiento adecuado a medida que varíen las exigencias.

1.10 PREGUNTAS DE REPASO

Las respuestas a las preguntas de repaso pueden hallarse en los apartados indicados.

- ¿Cuáles son las ventajas principales del empleo de SGBD para administrar los datos en aplicaciones que impliquen un amplio acceso a los datos? (**Apartados 1.1, 1.4**)
- ¿Cuándo se deben guardar los datos en SGBD en lugar de hacerlo en los archivos del sistema operativo y viceversa? (**Apartado 1.3**)
- ¿Qué son los modelos de datos? ¿Qué es el modelo de datos relacional? ¿Qué es la independencia con respecto a los datos y cómo la soportan los SGBD? (**Apartado 1.5**)
- Explíquense las ventajas de emplear un lenguaje de consultas para procesar los datos en lugar de los programas habituales. (**Apartado 1.6**)
- ¿Qué son las transacciones? ¿Qué garantías ofrecen los SGBD con respecto a las transacciones? (**Apartado 1.7**)
- ¿Qué son los bloqueos de los SGBD y por qué se emplean? ¿Qué es el registro previo a la escritura y por qué se utiliza? ¿Qué son los puntos de control y por qué se usan? (**Apartado 1.7**)
- Identifíquense los principales componentes de los SGBD y explíquese brevemente lo que hace cada uno de ellos. (**Apartado 1.8**)
- Explíquense los diferentes papeles de los administradores de bases de datos, los programadores de aplicaciones y los usuarios finales de las bases de datos. ¿Quién necesita saber más sobre los sistemas de bases de datos? (**Apartado 1.9**)

EJERCICIOS

Ejercicio 1.1 ¿Por qué elegir un sistema de bases de datos en lugar de limitarse a guardar los datos en los archivos del sistema operativo? ¿Cuándo tendría sentido *no* emplear un sistema de bases de datos?

22 Sistemas de gestión de bases de datos

Ejercicio 1.2 ¿Qué es la independencia lógica con respecto a los datos y por qué es importante?

Ejercicio 1.3 Explíquese la diferencia entre la independencia lógica con respecto a los datos y la física.

Ejercicio 1.4 Explíquense las diferencias entre los esquemas externo, interno y conceptual. ¿Cómo están relacionadas estas capas de esquemas con los conceptos de independencia lógica y física con respecto a los datos?

Ejercicio 1.5 ¿Cuáles son las responsabilidades de los DBA? Si se supone que el DBA no está interesado en ejecutar nunca sus propias consultas, ¿sigue necesitando comprender la optimización de las consultas? ¿Por qué?

Ejercicio 1.6 Avaro Puñocerrado desea guardar la información (nombres, direcciones, descripciones de momentos embarazosos, etc.) sobre las muchas víctimas de su lista. Como era de esperar, el volumen de datos le impulsa a comprar un sistema de bases de datos. Para ahorrar dinero, desea comprar uno con las mínimas características posibles, y piensa ejecutarlo como aplicación independiente en su ordenador personal. Por supuesto, Avaro no piensa compartir esa lista con nadie. Indíquese por cuáles de las siguientes características de los SGBD debe pagar Avaro; en cada caso, indíquese también el motivo de que Avaro deba (o no) pagar por esa característica del sistema que va a comprar.

1. Un dispositivo de seguridad.
2. Control de la concurrencia.
3. Recuperación de fallos.
4. Un mecanismo de vistas.
5. Un lenguaje de consultas.

Ejercicio 1.7 ¿Cuál de los elementos siguientes desempeña un papel importante en la *representación* de la información sobre el mundo real en las bases de datos? Explíquese brevemente.

1. El lenguaje de definición de datos.
2. El lenguaje de manipulación de datos.
3. El gestor de la memoria intermedia.
4. El modelo de datos.

Ejercicio 1.8 Descríbase la estructura de un SGBD. Si se actualiza el sistema operativo para que soporte alguna función nueva en los archivos del sistema operativo (por ejemplo, la posibilidad de obligar a que se guarde en disco alguna secuencia de bytes), ¿qué capa(s) del SGBD habría que volver a escribir para aprovechar esas funciones nuevas?

Ejercicio 1.9 Respóndanse las preguntas siguientes:

1. ¿Qué son las transacciones?
2. ¿Por qué los SGBD intercalan las acciones de las diferentes transacciones en lugar de ejecutar una transacción después de otra?
3. Qué deben garantizar los usuarios con respecto a las transacciones y a la consistencia de las bases de datos? ¿Qué deben garantizar los SGBD con respecto a la ejecución concurrente de varias transacciones y a la consistencia de las bases de datos?
4. Explíquese el protocolo de bloqueo estricto de dos fases.
5. ¿Qué es la propiedad WAL y por qué es importante?

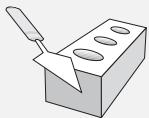
EJERCICIOS BASADOS EN PROYECTOS

Ejercicio 1.10 Empléese un navegador Web para examinar la documentación en HTML de Minibase. Inténtese captar la idea de su estructura global.

NOTAS BIBLIOGRÁFICAS

La evolución de los sistemas de gestión de bases de datos se describe en [189]. El empleo de modelos de datos para la descripción de los datos del mundo real se trata en [270], mientras que [271] contiene una taxonomía de los modelos de datos. Los tres niveles de abstracción se introdujeron en [118, 432]. El modelo de datos en red se describe en [118], mientras que [472] trata de varios sistemas comerciales basados en ese modelo. [438] contiene un buen conjunto comentado de trabajos orientados a los sistemas sobre la gestión de las bases de datos.

Entre los textos que tratan de los sistemas de gestión de bases de datos están [137, 157, 199, 220, 298, 347, 414, 455, 463]. En [136] se ofrece un estudio detallado del modelo relacional desde un punto de vista conceptual y es destacable por su amplia bibliografía comentada. [346] presenta una perspectiva orientada al rendimiento con referencias a varios sistemas comerciales. En [157] y [414] se ofrece una amplia cobertura de los conceptos de los sistemas de bases de datos, incluido un estudio de los modelos de datos jerárquico y de red. [220] pone el énfasis en la conexión entre los lenguajes de consultas para bases de datos y la programación lógica. [463] se centra en los modelos de datos. De estos textos, [455] ofrece el tratamiento más detallado de los aspectos teóricos. Entre los textos dedicados a los aspectos teóricos están [1, 29, 309]. El libro [452] incluye un apartado sobre bases de datos que contiene artículos de investigación introductorios sobre diversos temas.



2

INTRODUCCIÓN AL DISEÑO DE BASES DE DATOS

- ➔ ¿Qué pasos se siguen en el diseño de bases de datos?
- ➔ ¿Por qué se utiliza el modelo ER para crear el diseño inicial?
- ➔ ¿Cuáles son los conceptos principales del modelo ER?
- ➔ ¿Cuáles son las directrices para un empleo eficaz del modelo ER?
- ➔ ¿Cómo encaja el diseño de bases de datos en el marco del diseño global de software complejo en las grandes empresas?
- ➔ ¿Qué es UML y qué relación tiene con el modelo ER?
- ➡ **Conceptos fundamentales:** diseño de bases de datos, diseño conceptual, lógico y físico; modelo entidad-relación (ER), conjunto de entidades, conjunto de relaciones, atributo, ejemplar, clave; restricciones de integridad, relaciones de una a varias y de varias a varias, restricciones de participación; entidades débiles, jerarquías de clases, agregación; UML, diagramas de clases, diagramas de bases de datos, diagramas de componentes.

Los grandes hombres de éxito del mundo han empleado su imaginación. Se adelantan en su pensamiento, crean su propia imagen mental y luego trabajan para materializar esa imagen en todos sus detalles, rellenando aquí, añadiendo un poco allá, alterando esa pizca y aquélla, pero trabajando sin pausa, construyendo sin pausa.

— Robert Collier

El *modelo de datos entidad-relación (ER)* permite describir los datos implicados en empresas reales en términos de objetos y de sus relaciones, y se emplea mucho para desarrollar el diseño preliminar de las bases de datos. Aporta conceptos útiles que permiten pasar de una descripción informal de lo que los usuarios desean de su base de datos a otra más detallada y precisa que se pueda implementar en un SGBD. En este capítulo se presenta el modelo ER

y se estudia el modo en que sus características permiten modelar fielmente una amplia gama de datos.

Se comenzará con una introducción al diseño de bases de datos en el Apartado 2.1 con objeto de motivar el estudio del modelo ER. Dentro del contexto más general del proceso global de diseño, el modelo ER se emplea en la fase denominada *diseño conceptual de bases de datos*. Luego se introducirá el modelo ER en los apartados 2.2, 2.3 y 2.4. En el Apartado 2.5 se estudian aspectos del diseño de bases de datos que afectan al modelo ER. El diseño conceptual de bases de datos para grandes empresas se trata brevemente en el Apartado 2.6. En el Apartado 2.7 se presenta una visión general de UML, un enfoque de diseño y modelado de ámbito más general que el modelo ER.

En el Apartado 2.8 se presenta el estudio de un caso que se emplea como ejemplo a lo largo de todo el libro. El caso que se estudia es el diseño de principio a fin de una base de datos para una tienda en Internet. Se ilustran los dos primeros pasos del diseño de bases de datos (análisis de requisitos y diseño conceptual) en el Apartado 2.8. En capítulos posteriores se amplía este caso de estudio para tratar el resto de pasos del proceso de diseño.

Es de destacar que se han utilizado muchas variaciones de los diagramas ER y que no se ha impuesto ninguna norma generalmente aceptada. La presentación de este capítulo es representativa de la familia de modelos ER e incluye una selección de las características más populares.

2.1 DISEÑO DE BASES DE DATOS Y DIAGRAMAS ER

Se comienza el estudio del diseño de bases de datos mediante la observación de que no suele tratarse más que de una parte, aunque fundamental para las aplicaciones que hacen amplio uso de los datos, del diseño de sistemas de software de mayor alcance. No obstante, el principal centro de atención de este libro es el diseño de la base de datos, y sólo se tratarán por encima otros aspectos del diseño de software. Este punto se vuelve a tratar en el Apartado 2.7.

El proceso de diseño de bases de datos puede dividirse en seis etapas. El modelo ER es muy relevante para los tres primeros pasos.

1. **Análisis de requisitos.** El primer paso del diseño de aplicaciones de bases de datos es comprender los datos que se deben guardar en la base de datos, las aplicaciones que se deben construir sobre ellos y las operaciones que son más frecuentes e imponen requisitos de rendimiento. En otras palabras, hay que averiguar lo que los usuarios esperan de la base de datos. Se trata normalmente de un proceso informal que supone discusiones con grupos de usuarios, un estudio del entorno operativo en vigor y del modo en que se espera que éste cambie, el análisis de toda la documentación disponible sobre las aplicaciones existentes que se espera que la base de datos sustituya o complemente, etc. Se han propuesto varias metodologías para la organización y presentación de la información recogida en esta etapa, y se han desarrollado algunas herramientas automatizadas para soportar el proceso.
2. **Diseño conceptual de bases de datos.** La información reunida en el análisis de requisitos se emplea para desarrollar una descripción de alto nivel de los datos que se van a guardar en la base de datos, junto con las restricciones que se sabe que se impondrán sobre esos datos. Este paso se suele llevar a cabo empleando el modelo ER y se discute en el resto

Herramientas para el diseño de bases de datos. Los fabricantes de SGBDR y otros fabricantes ofrecen herramientas de diseño. Por ejemplo, véase el siguiente enlace para conocer más detalles sobre las herramientas de diseño y análisis de Sybase:

http://www.sybase.com/products/application_tools

El siguiente ofrece detalles sobre las herramientas de Oracle:

<http://www.oracle.com/tools>

de este capítulo. El modelo ER es uno de los modelos de datos de alto nivel, o **semánticos**, empleados en el diseño de bases de datos. El objetivo es crear una descripción sencilla de los datos que se acerque mucho a lo que los usuarios y los desarrolladores piensan de los datos (y de la gente y de los procesos que se van a representar con esos datos). Esto facilita la discusión entre todas las personas implicadas en el proceso de diseño, aunque no tengan formación técnica. Al mismo tiempo, el diseño inicial debe ser lo bastante preciso como para permitir una traducción directa a un modelo de datos soportado por algún sistema comercial de bases de datos (lo que, en la práctica, significa el modelo relacional).

3. **Diseño lógico de bases de datos.** Hay que escoger un SGBD que implemente nuestro diseño de la base de datos y transformar el diseño conceptual de la base de datos en un esquema de base de datos del modelo de datos del SGBD elegido. Sólo se considerarán SGBD relacionales y, por tanto, la tarea en la etapa de diseño lógico es convertir el esquema ER en un esquema de base de datos relacional. Este paso se examinará con detalle en el Capítulo 3; el resultado es un esquema conceptual, a veces denominado **esquema lógico**, del modelo de datos relacional.

2.1.1 Más allá del diseño ER

El diagrama ER no es más que una descripción aproximada de los datos creada mediante la evaluación subjetiva de la información reunida durante el análisis de requisitos. A menudo, un análisis más detenido puede refinar el esquema lógico obtenido al final del Paso 3. Una vez se dispone de un buen esquema lógico hay que tomar en consideración los criterios de rendimiento y diseñar el esquema físico. Finalmente, hay que abordar los aspectos de seguridad y garantizar que los usuarios puedan tener acceso a los datos que necesiten, pero no a los que se les desee ocultar. Las tres etapas restantes del diseño de bases de datos se describen brevemente a continuación:

4. **Refinamiento de los esquemas.** El cuarto paso del diseño de bases de datos es el análisis del conjunto de relaciones del esquema relacional de la base de datos para identificar posibles problemas y refinarlo. A diferencia de los pasos de análisis de requisitos y de diseño conceptual, que son esencialmente subjetivos, el refinamiento del esquema se puede guiar por una teoría elegante y potente. Esta teoría de *normalización* de relaciones —reestructurarlas para garantizar propiedades deseables— se estudia en el Capítulo 12.

5. **Diseño físico de bases de datos.** En este paso se toman en consideración las cargas de trabajo típicas esperadas que deberá soportar la base de datos y se refinará aún más el diseño de la base de datos para garantizar que cumple los criterios de rendimiento deseados. Puede que este paso no implique más que la creación de índices para algunas tablas y el agrupamiento de otras, o puede que suponga un rediseño sustancial de partes del esquema de la base de datos obtenido de los pasos de diseño anteriores. El diseño físico y el ajuste de las bases de datos se estudia en el Capítulo 13.
6. **Diseño de aplicaciones y de la seguridad.** Cualquier proyecto de software que implique a una base de datos debe tomar en consideración aspectos de su aplicación que van más allá de la propia base de datos. Las metodologías de diseño como UML (Apartado 2.7) intentan abordar todo el ciclo de diseño y desarrollo del software. En resumen, hay que identificar las entidades (por ejemplo, los usuarios, los grupos de usuarios, los departamentos) y los procesos relacionados con la aplicación. Hay que describir el papel de cada entidad en cada proceso que se refleje en una tarea de la aplicación, como parte del flujo de trabajo completo de esa tarea. Para cada papel hay que identificar las partes de la base de datos que debe tener accesibles y las que *no* debe tener accesibles, y adoptar las medidas necesarias para que esas reglas de acceso se cumplan. Los SGBD ofrecen varios mecanismos para ayudar en este paso, y se tratan en el Capítulo 14.

En la fase de implementación hay que codificar cada tarea en un lenguaje de programación (por ejemplo, Java) y emplear el SGBD para tener acceso a los datos. El desarrollo de aplicaciones se estudia en los Capítulos 6 y 7.

En general, la división del proceso de diseños en varios pasos debería verse como una clasificación de los *tipos* de pasos relacionados con el diseño. De manera realista, aunque se podría comenzar con el proceso en seis etapas que se ha esbozado aquí, el diseño completo de una base de datos probablemente necesite una fase posterior de **ajuste** en la que los seis tipos de etapas del diseño se entrelacen y se repitan hasta que el diseño sea satisfactorio.

2.2 ENTIDADES, ATRIBUTOS Y CONJUNTOS DE ENTIDADES

Una **entidad** es un objeto del mundo real que puede distinguirse de otros objetos. Ejemplos de entidades son: el juguete Dragón verde, el departamento de juguetes, el responsable del departamento de juguetes o la dirección postal del responsable del departamento de juguetes. Suele resultar útil identificar conjuntos de entidades similares. Un conjunto así se denomina **conjunto de entidades**. Obsérvese que no hace falta que los conjuntos de entidades sean disjuntos; tanto el conjunto de empleados del departamento de juguetes como el conjunto de empleados del departamento de electrodomésticos pueden contener al empleado Juanjo Díaz (que resulta que trabaja en los dos departamentos). También se podría definir un conjunto de entidades denominado Empleados que contuviera tanto al conjunto de empleados del departamento de juguetes como al de los empleados del departamento de electrodomésticos.

Cada entidad se describe empleando un conjunto de **atributos**. Todas las entidades de un conjunto de entidades dado tienen los mismos atributos; eso es lo que quiere decir *similar*. (Esta afirmación es una simplificación excesiva, como se verá cuando se discutan las jerarquías

de herencia en el Apartado 2.4.4, pero resulta suficiente por ahora y hace destacar la idea principal.) La selección de los atributos refleja el nivel de detalle con el que se desea representar la información relativa a las entidades. Por ejemplo, el conjunto de entidades Empleados podría utilizar el nombre, el número del documento nacional de identidad (DNI) y el número de la plaza de aparcamiento de cada empleado. Sin embargo, no se guardará, por ejemplo, la dirección del empleado (ni su sexo ni su edad).

Para cada atributo asociado con un conjunto de entidades hay que identificar el **dominio** de valores posibles. Por ejemplo, el dominio asociado con el atributo *nombre* de Empleados podría ser el conjunto de cadenas de caracteres de longitud veinte¹. Como ejemplo adicional, si la empresa califica a sus empleados según una escala del uno al diez y guarda las calificaciones en un campo denominado *calificaciones*, el dominio asociado consistirá en los enteros del uno al diez. Además, para cada conjunto de entidades, se escoge una *clave*. Una **clave** es un conjunto mínimo de atributos cuyos valores identifican de manera única a cada entidad del conjunto. Puede haber más de una clave **candidata**; en ese caso, se escogerá una de ellas como clave **principal**. Por ahora se supondrá que cada conjunto de entidades contiene, como mínimo, un conjunto de atributos que identifica de manera única a cada entidad de ese conjunto; es decir, que el conjunto de entidades contiene una clave. Este punto se volverá a tratar en el Apartado 2.4.3.

El conjunto de entidades Empleados con los atributos *dni*, *nombre* y *plaza* se muestra en la Figura 2.1. Cada conjunto de entidades se representa por un rectángulo, y cada atributo por un óvalo. Los atributos de la clave principal están subrayados. Se podría indicar la información sobre el dominio junto con el nombre de cada atributo, pero se ha omitido para que las figuras sean más concisas. La clave es *dni*.

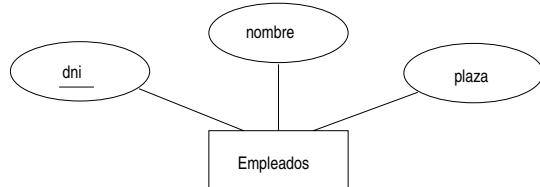


Figura 2.1 El conjunto de entidades Empleados

2.3 LAS RELACIONES Y LOS CONJUNTOS DE RELACIONES

Una **relación** es una asociación entre dos o más entidades. Por ejemplo, puede que se tenga una relación en que Avelino trabaja en el departamento farmacéutico. Al igual que con las entidades, puede que se desee reunir un conjunto de relaciones similares en un **conjunto de**

¹Para evitar confusiones se da por supuesto que los nombres de los atributos no se repiten en los conjuntos de entidades. Esto no supone una verdadera limitación, ya que siempre se puede utilizar el nombre del conjunto de entidades para resolver las ambigüedades si el mismo nombre de atributo se utiliza en más de un conjunto de entidades.

entidades. Se puede considerar a los conjuntos de relaciones como conjuntos de n -tuplas:

$$\{(e_1, \dots, e_n) \mid e_1 \in E_1, \dots, e_n \in E_n\}$$

Cada n -tupla denota una relación que implica a n entidades, e_1 a e_n , donde la entidad e_i se halla en el conjunto de entidades E_i . En la Figura 2.2 puede verse el conjunto de relaciones Trabaja_en, en la que cada relación indica un departamento en el que trabaja ese empleado. Obsérvese que puede que varios conjuntos de relaciones impliquen a los mismos conjuntos de entidades. Por ejemplo, también se podría tener un conjunto de relaciones Dirige que implique a Empleados y Departamentos.

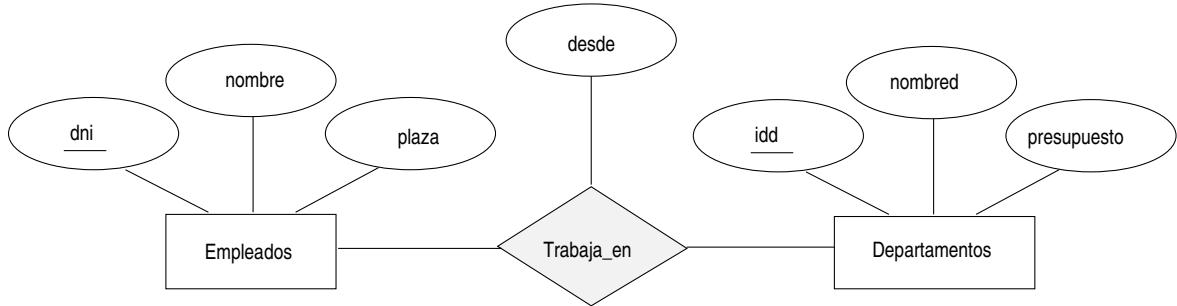


Figura 2.2 El conjunto de relaciones Trabaja_en

Las relaciones también pueden tener **atributos descriptivos**. Los atributos descriptivos se emplean para registrar la información sobre la relación, más que sobre las entidades participantes; por ejemplo, puede que se deseé registrar que Avelino trabaja en el departamento farmaceútico desde enero de 1991. Esta información se captura en la Figura 2.2 añadiendo un atributo, *desde*, a Trabaja_en. Cada relación debe identificarse de manera única por sus entidades participantes, sin necesidad de referencia alguna a los atributos descriptivos. En el conjunto de relaciones Trabaja_en, por ejemplo, cada relación Trabaja_en debe quedar identificada de manera única por la combinación de empleado *dni* y departamento *idd*. Por tanto, para una pareja empleado-departamento dada, no se puede tener más de un valor *desde* asociado.

Cada **ejemplar** de un conjunto de relaciones es un conjunto de relaciones. De manera intuitiva se puede considerar cada ejemplar como una “instantánea” del conjunto de relaciones en un momento temporal dado. Puede verse un ejemplar del conjunto de relaciones Trabaja_en en la Figura 2.3. Cada entidad Empleados se denota por su *dni*, y cada entidad Departamentos se denota por su *idd*, en aras de la simplicidad. El valor *desde* se muestra junto a cada relación. (Los comentarios “varias a varias” y “participación total” de la figura se explican más adelante, cuando se estudian las restricciones de integridad.)

Como ejemplo adicional de diagrama ER, supóngase que cada departamento tiene oficinas en varias ubicaciones y que se desea registrar las ubicaciones en las que trabaja cada empleado. Esta relación es **ternaria**, ya que hay que registrar la asociación entre cada empleado, el departamento y la ubicación. El diagrama ER de esta variante de Trabaja_en, que se denominará Trabaja_en2, puede verse en la Figura 2.4.

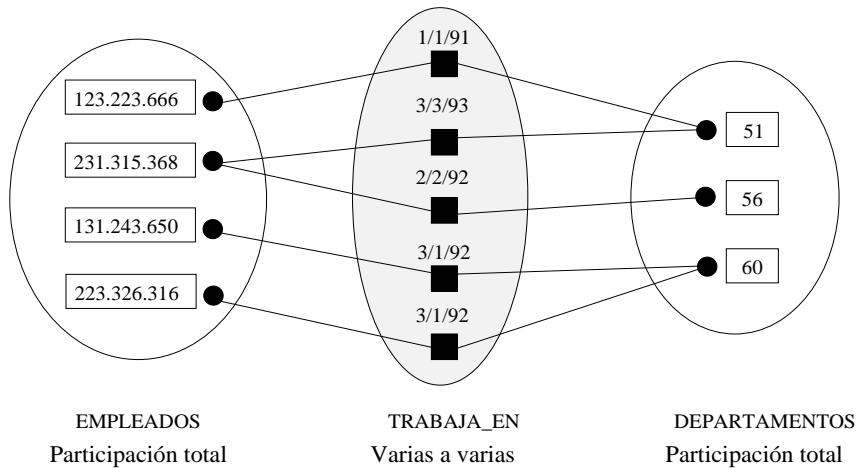


Figura 2.3 Un ejemplar del conjunto de relaciones Trabaja_en

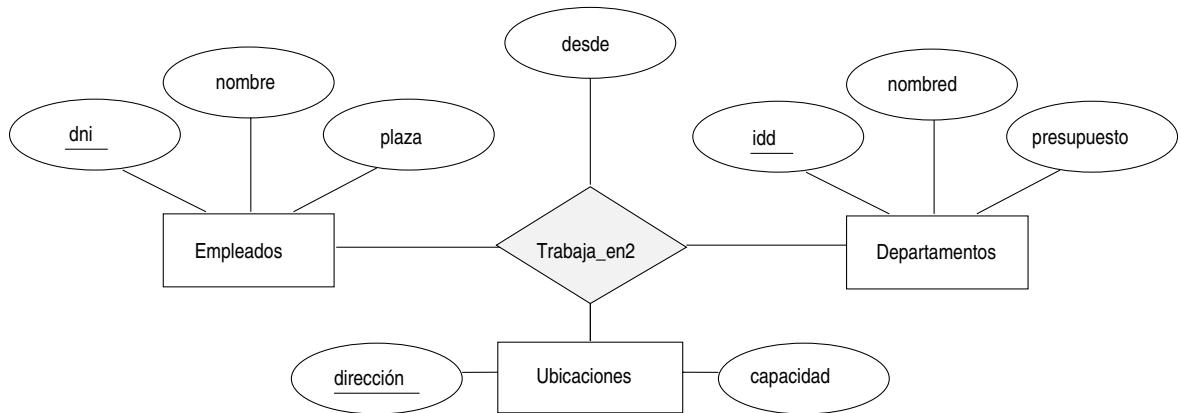


Figura 2.4 Un conjunto de relaciones ternarias

No hace falta que los conjuntos de entidades que participan en un conjunto de relaciones sean distintos; puede que a veces una relación implique a dos entidades del mismo conjunto de entidades. Por ejemplo, considérese el conjunto de relaciones Informa_a que puede verse en la Figura 2.5. Como los empleados rinden cuentas a otros empleados, las relaciones de Informa_a son de la forma (emp_1, emp_2) , donde tanto emp_1 como emp_2 son entidades de Empleados. Sin embargo, interpretan **papeles** diferentes: emp_1 rinde cuentas al empleado encargado emp_2 , lo que se refleja en los **indicadores de papeles supervisor y subordinado** en la Figura 2.5. Si un conjunto de entidades desempeña más de un papel, el indicador de papeles concatenado con un nombre de atributo del conjunto de entidades da un nombre único para cada atributo del conjunto de relaciones. Por ejemplo, el conjunto de relaciones Informa_a tiene los atributos correspondientes al *dni* del supervisor y al *dni* del subordinado, y el nombre de esos atributos es *dni_supervisor* y *dni_subordinado*.

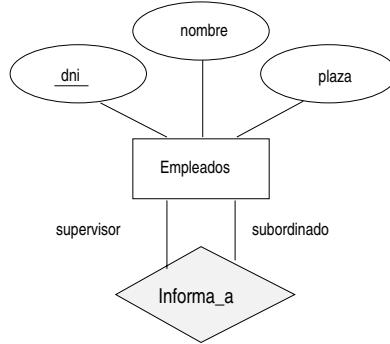


Figura 2.5 El conjunto de relaciones Informa_a

2.4 OTRAS CARACTERÍSTICAS DEL MODELO ER

A continuación se examinarán algunas de las estructuras del modelo ER que permiten describir algunas propiedades sutiles de los datos. La expresividad del modelo ER es una razón importante de su amplia utilización.

2.4.1 Restricciones de clave en relaciones

Considérese la relación Trabaja_en de la Figura 2.2. Cada empleado puede trabajar en varios departamentos, y cada departamento puede tener varios empleados, como puede verse en la Figura 2.3. El empleado 231.315.368 ha trabajado en el Departamento 51 desde 3/3/93 y en el Departamento 56 desde 2/2/92. El Departamento 51 tiene dos empleados.

Considérese ahora otro conjunto de relaciones denominado Dirige entre los conjuntos de entidades Empleados y Departamentos tal que cada departamento tenga como máximo un encargado, aunque se permite que cada empleado dirija más de un departamento. La restricción de que cada departamento tenga como máximo un encargado es un ejemplo de **restricción de clave**, e implica que cada entidad de departamentos aparezca como máximo en una relación

Dirige en cualquier ejemplar admisible de Dirige. Esta restricción se indica en el diagrama ER de la Figura 2.6 mediante una flecha que va de Departamentos a Dirige. De manera intuitiva, la flecha indica que, dada una entidad Departamentos, se puede determinar de manera unívoca la relación Dirige en la que aparece.

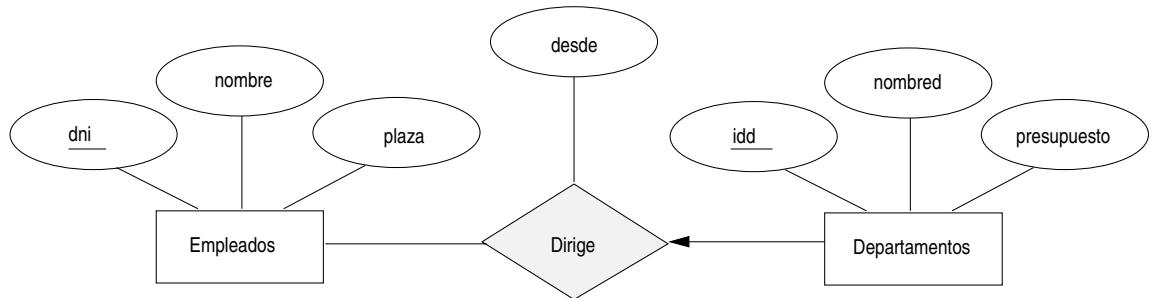


Figura 2.6 Restricción de clave en Dirige

En la Figura 2.7 se muestra un ejemplar del conjunto de relaciones Dirige. Aunque se trate también de un posible ejemplar del conjunto de relaciones Trabaja_en, el ejemplar de Trabaja_en que puede verse en la Figura 2.3 viola la restricción de clave en Dirige.

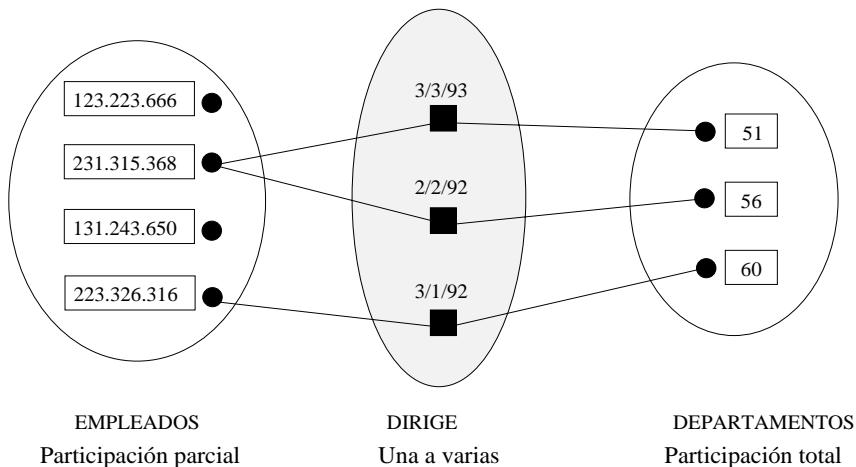


Figura 2.7 Ejemplar del conjunto de relaciones Dirige

Se dice a veces que los conjuntos de relaciones como Dirige son de **una a varias**, para indicar que *cada* empleado se puede asociar con *varios* departamentos (en funciones de encargado), mientras que cada departamento se puede asociar, como máximo, con un empleado como encargado. Por el contrario, se dice que el conjunto de relaciones Trabaja_en, en el que se permite que cada empleado trabaje en varios departamentos y que cada departamento tenga varios empleados, es de **varias a varias**.

Si se añade al conjunto de relaciones Dirige la restricción de que cada empleado pueda dirigir, como máximo, un departamento, lo que se indicaría añadiendo una flecha de Empleados a Dirige en la Figura 2.6, se tendrá un conjunto de relaciones de **una a una**.

Restricciones de clave en relaciones ternarias

Se puede ampliar este convenio —y el concepto subyacente de restricción de clave— a los conjuntos de relaciones que abarcan tres o más conjuntos de entidades: si el conjunto de entidades E tiene una restricción de clave en el conjunto de relaciones R, cada entidad de un ejemplar concreto de E aparecerá, como máximo, en una relación de (el ejemplar correspondiente de) R. Para indicar una restricción de clave sobre el conjunto de entidades E del conjunto de relaciones R, se traza una flecha de E a R.

En la Figura 2.8 se muestra una relación ternaria con una restricción de clave. Cada empleado trabaja, como máximo, en un departamento y en una única ubicación. Puede verse un ejemplar del conjunto de relaciones Trabaja_en3 en la Figura 2.9. Téngase en cuenta que cada departamento puede asociarse con varios empleados y ubicaciones y que cada ubicación puede asociarse con varios departamentos y empleados; sin embargo, cada empleado está asociado con un solo departamento y una única ubicación.

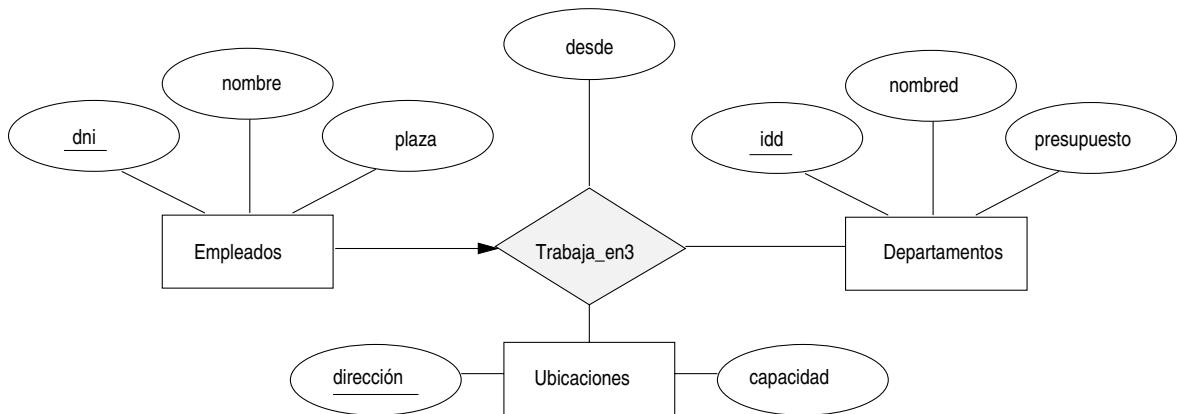


Figura 2.8 Un conjunto de relaciones ternarias con restricción de clave

2.4.2 Restricciones de participación

La restricción de clave sobre Dirige indica que cada departamento tiene, como máximo, un encargado. Una pregunta que resulta lógico formularse es si todos los departamentos tienen encargado. Supongamos que se exige que cada departamento tenga un encargado. Este requisito es un ejemplo de **restricción de participación**; se dice que la participación del conjunto de entidades Departamentos en el conjunto de relaciones Dirige es **total**. Una participación que no es total se dice de que es **parcial**. A modo de ejemplo, la participación del conjunto de entidades Empleados en Dirige es parcial, ya que no todos los empleados consiguen dirigir un departamento.

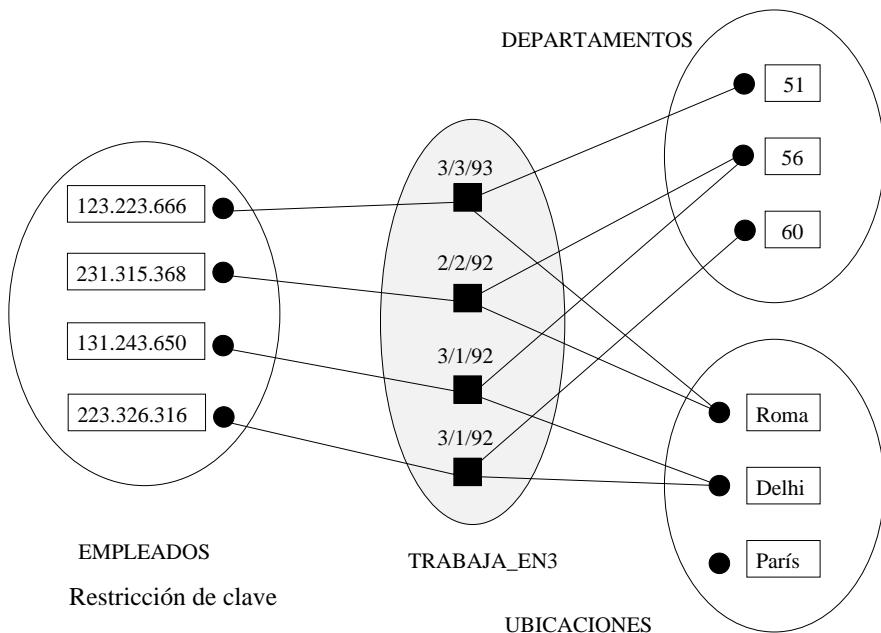


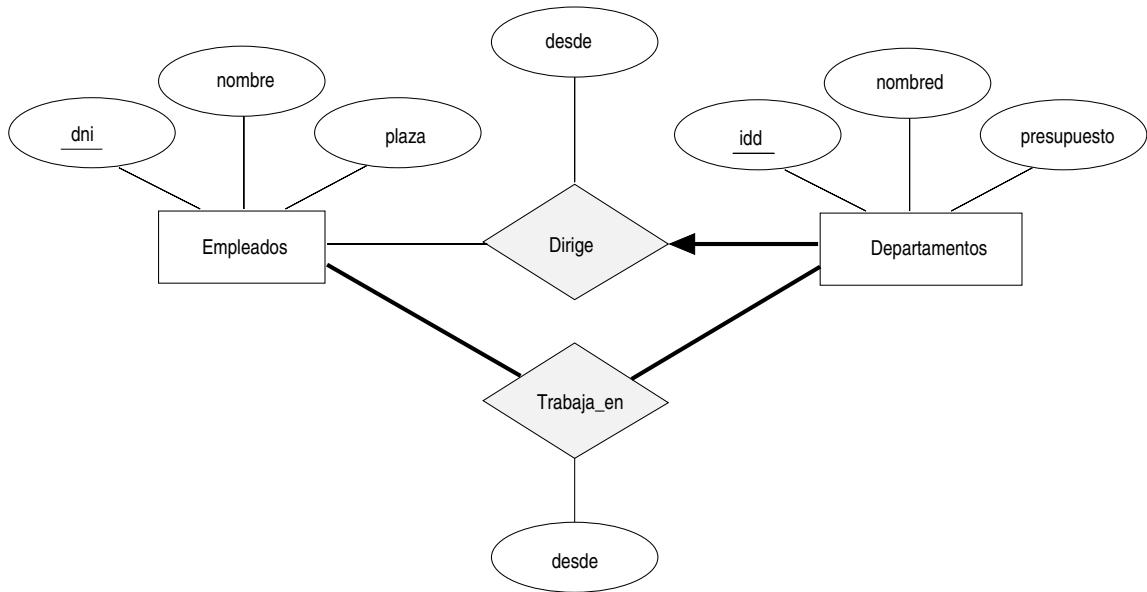
Figura 2.9 Un ejemplar de Trabaja_en3

De vuelta al conjunto de relaciones Trabaja_en, resulta natural esperar que cada empleado trabaje, como mínimo, en un departamento y que cada departamento tenga, como mínimo, un empleado. Esto significa que tanto la participación de Empleados como la de Departamentos en Trabaja_en es total. El diagrama ER de la Figura 2.10 muestra tanto al conjunto de relaciones Dirige como al conjunto de relaciones Trabaja_en y a todas las restricciones dadas. Si la participación de un conjunto de entidades en un conjunto de relaciones es total, ambos se conectan mediante una línea gruesa; de manera independiente, la presencia de una flecha indica una restricción de clave. Los ejemplares de Trabaja_en y Dirige mostrados en las Figuras 2.3 y 2.7 satisfacen todas las restricciones de la Figura 2.10.

2.4.3 Entidades débiles

Hasta ahora se ha supuesto que entre los atributos asociados a un conjunto de entidades se incluye una clave. Esta suposición no siempre se cumple. Por ejemplo, supóngase que los empleados pueden suscribir pólizas de seguros que cubran a las personas que dependen de ellos. Se desea registrar información sobre esas pólizas, incluyendo a las personas cubiertas en cada póliza; pero esa información es, en realidad, lo único que interesa de las personas que dependen de cada empleado. Si un empleado deja de serlo, las pólizas que hubiera suscrito se cancelan, y se desea eliminar de la base de datos toda la información relevante sobre esas pólizas y sobre las personas que dependen de ese antiguo empleado.

Se podría decidir identificar en este caso a cada persona que depende de un empleado únicamente por su nombre, ya que es razonable esperar que todas las personas que dependen de un empleado tengan nombres diferentes. Por tanto, los atributos del conjunto de entidades

**Figura 2.10** Dirige y Trabaja_en

Beneficiarios podrían ser *nombrrep* y *edad*. El atributo *nombrrep* no identifica de manera única a cada persona que depende de un empleado. Recuérdese que la clave para Empleados es *dni*; por tanto, se podrían tener dos empleados llamados Sánchez con un hijo llamado José.

Beneficiarios es un ejemplo de **conjunto de entidades débiles**. Cada entidad débil sólo se puede identificar de manera única tomando en consideración alguno de sus atributos junto con la clave principal de otra entidad, que se conoce como **proprietaria identificadora**.

Se deben cumplir las restricciones siguientes:

- El conjunto de entidades propietario y el conjunto de entidades débiles deben participar en un conjunto de relaciones de una a varias (cada entidad propietaria se asocia con una o varias entidades débiles, pero cada entidad débil sólo tiene una propietaria). Este conjunto de relaciones se denomina **conjunto de relaciones identificadoras** del conjunto de entidades débiles.
- El conjunto de entidades débiles debe tener participación total en el conjunto de relaciones identificadoras.

Por ejemplo, cada entidad de Beneficiarios sólo se puede identificar de manera única si se toma la clave de la entidad Empleados *proprietaria* y *nombrrep* de la entidad Beneficiarios. El conjunto de atributos de un conjunto de entidades débiles que identifica de manera única a una entidad débil para una entidad propietaria dada se denomina *clave parcial* del conjunto de entidades débiles. En nuestro ejemplo, *nombrrep* es una clave parcial de Beneficiarios.

El conjunto de entidades débiles Beneficiarios y su relación con Empleados se muestra en la Figura 2.11. La participación total de Beneficiarios en Póliza se indica enlazándolos mediante una línea gruesa. La flecha que va de Beneficiarios a Póliza indica que cada entidad

de Beneficiarios aparece, como máximo, en una relación de Póliza (en realidad, exactamente en una, debido a la restricción de participación). Para subrayar el hecho de que Beneficiarios es una entidad débil y Póliza es su relación identificadora se dibujan las dos con líneas oscuras. Para indicar que *nombrep* es una clave parcial de Beneficiarios, se subraya con una línea discontinua. Esto significa que puede haber perfectamente dos personas que dependan de empleados y tengan el mismo valor de *nombrep*.

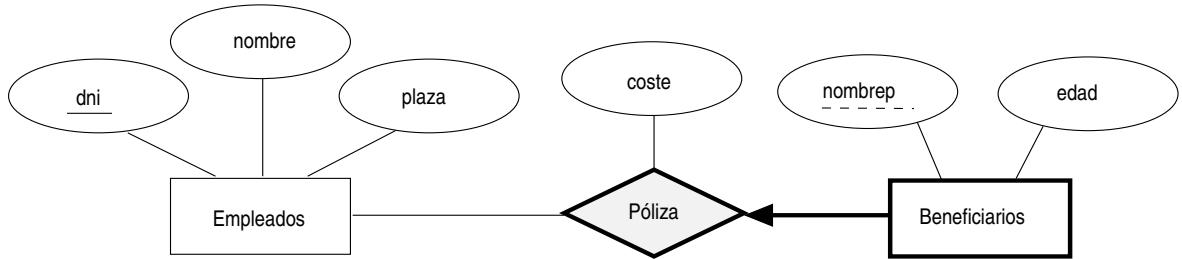


Figura 2.11 Un conjunto de entidades débiles

2.4.4 Jerarquías de clases

A veces resulta natural clasificar las entidades en un conjunto de entidades en subclases. Por ejemplo, puede que se desee hablar del conjunto de entidades Empleados_temp y del conjunto de entidades Empleados_fijos para distinguir el modo en que se calcula su sueldo. Puede que se hayan definido los atributos *horas_trabajadas* y *sueldo_hora* definidos para Empleados_temp y el atributo *idcontrato* para Empleados_fijos.

Se desea que todas las entidades de cada uno de esos conjuntos sean también entidades de Empleados y, como tales, deberán tener definidos todos los atributos de empleados. Por tanto, los atributos definidos para una entidad Empleados_temp dada son los atributos de Empleados más los de Empleados_temp. Se dice que los atributos del conjunto de entidades Empleados se **heredan** por el conjunto de entidades Empleados_temp y que una entidad de Empleados_temp **ES** una entidad de Empleados. Además —y a diferencia de las jerarquías de clases de los lenguajes de programación como C++— hay una restricción para las consultas sobre los ejemplares de estos conjuntos de entidades: las consultas que pidan todas las entidades Empleados también deben tomar en consideración las entidades Empleados_temp y Empleados_fijos. La Figura 2.12 ilustra la jerarquía de clases.

El conjunto de entidades Empleados también se puede clasificar según un criterio diferente. Por ejemplo, se puede identificar un subconjunto de empleados como Empleados_veteranos. Se puede modificar la Figura 2.12 para que refleje esta modificación añadiendo un segundo nodo ES como hijo de Empleados y haciendo a Empleados_veteranos hijo de ese nodo. Se puede seguir clasificando cada uno de esos conjuntos de entidades y crear una jerarquía ES multinivel.

Las jerarquías de clases se pueden considerar desde dos puntos de vista:

- Empleados está **especializada** en subclases. La especialización es el proceso de identificación de subconjuntos de un conjunto de entidades dado (la **superclase**) que comparten

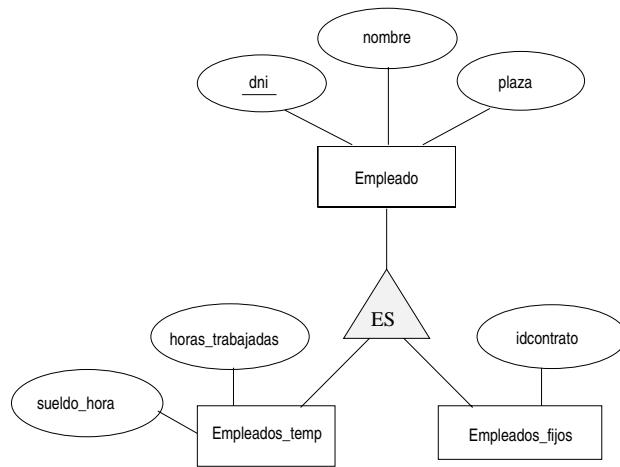


Figura 2.12 Jerarquía de clases

alguna característica distintiva. Generalmente se define en primer lugar la superclase, a continuación se definen las subclases y luego se añaden los atributos y los conjuntos de relaciones específicos de cada subclase.

- Empleados_temp y Empleados_fijos se **generalizan** mediante Empleados. Como ejemplo adicional, se pueden generalizar los conjuntos de entidades Motoras y Coches en el conjunto de entidades Vehículos_motorizados. La generalización consiste en identificar alguna característica común de un conjunto de conjuntos de entidades y crear un nuevo conjunto de entidades que contenga entidades que posean esas características comunes. Generalmente se definen en primer lugar las subclases, a continuación se define la superclase y luego se definen los conjuntos de relaciones que implican a la superclase.

Se pueden especificar dos tipos de restricciones con respecto a las jerarquías ES: las restricciones de *solapamiento* y de *cobertura*. Las **restricciones de solapamiento** determinan si se permite que dos clases contengan la misma entidad. Por ejemplo, ¿puede Avelino ser una entidad Empleados_temp y una entidad Empleados_fijos a la vez? De manera intuitiva, no. ¿Puede ser a la vez una entidad Empleados_fijos y una entidad Empleados_veteranos? De manera intuitiva, sí. Esto se denota escribiendo “Empleados_fijos SOLAPA A Empleados_Veteranos”. En ausencia de una afirmación de este tipo, se da por supuesto de manera predeterminada que se restringe a los conjuntos de entidades a no solaparse.

Las **restricciones de cobertura** determinan si las entidades de las subclases incluyen de manera colectiva a todas las entidades de la superclase. Por ejemplo, ¿tienen que pertenecer todas las entidades Empleados a alguna de sus subclases? De manera intuitiva, no. ¿Tienen que ser todas las entidades Vehículos_motorizados una entidad Motoras o una entidad Coches? De manera intuitiva, sí; una propiedad característica de las jerarquías de generalización es que todos los ejemplares de una superclase son, a su vez, ejemplares de una subclase. Esto se denota escribiendo “Motoras Y Coches CUBREN Vehículos_motorizados”. En ausencia de una afirmación de este tipo se supone de manera predeterminada que no hay ninguna restricción de cobertura; se pueden tener vehículos de motor que no sean ni motoras ni coches.

Hay dos motivos básicos para identificar subclases (por especialización o por generalización):

1. Puede que se desee añadir atributos descriptivos que sólo tengan sentido para las entidades de una subclase dada. Por ejemplo, *sueldo hora* no tiene sentido para una entidad *Empleados_fijos*, cuya paga se determina mediante un contrato individual.
2. Puede que se desee identificar el conjunto de entidades que participan en una relación dada. Por ejemplo, puede que se desee definir la relación *Dirige* de modo que los conjuntos de entidades participantes sean *Empleados_veteranos* y *Departamentos*, para garantizar que sólo los empleados veteranos puedan ser encargados. Como ejemplo adicional, puede que *Motoras* y *Coches* tengan atributos descriptivos diferentes (por ejemplo, tonelaje y número de puertas) pero, como entidades de *Vehículos_motorizados*, deben estar matriculados. La información sobre la matrícula se puede capturar mediante una relación *Matriculado_por* entre *Vehículos_motorizados* y un conjunto de entidades denominado *Propietarios*.

2.4.5 Agregación

Como se ha definido hasta ahora, un conjunto de relaciones es una asociación entre conjuntos de entidades. A veces hay que modelar las relaciones entre un conjunto de entidades y de *relaciones*. Supóngase que se tiene un conjunto de entidades denominado *Proyectos* y que cada entidad de *Proyectos* está patrocinada por uno o varios departamentos. El conjunto de relaciones *Patrocina* capture esa información. El departamento que patrocina un proyecto puede asignar empleados para que controlen el patrocinio. De manera intuitiva, *Controla* debería ser un conjunto de relaciones que asocie relaciones *Patrocina* (en vez de entidades *Proyectos* o *Departamentos*) con entidades *Empleados*. Sin embargo, las relaciones que se han definido asocian dos o más *entidades*, no relaciones.

Para definir un conjunto de relaciones como *Controla* se introduce una nueva característica del modelo ER, denominada *agregación*. La **agregación** permite indicar que un conjunto de relaciones (identificado mediante un cuadro discontinuo) participa en otro conjunto de relaciones. Esto se ilustra en la Figura 2.13, con un cuadro discontinuo alrededor de *Patrocina* (y sus conjuntos de entidades participantes) para denotar la agregación. Esto permite tratar de manera efectiva *Patrocina* como un conjunto de entidades a los efectos de definir el conjunto de relaciones *Controla*.

¿Cuándo se debe emplear la agregación? De manera intuitiva se emplea cuando hace falta expresar una relación entre relaciones. ¿Pero no se pueden expresar relaciones que impliquen a otras relaciones sin emplear la agregación? En el ejemplo anterior, ¿por qué no hacer de *Patrocina* una relación ternaria? La respuesta es que realmente hay dos relaciones diferentes, *Patrocina* y *Controla*, cada una de las cuales tendrá sus propios atributos. Por ejemplo, la relación *Controla* tiene el atributo *hasta* que registra la fecha hasta la que un empleado ha sido nombrado controlador del patrocinio. Compárese este atributo con el atributo *desde* de *Patrocina*, que es la fecha en que el patrocinio entró en vigor. El empleo de la agregación en lugar de una relación ternaria también puede deberse a determinadas restricciones de integridad, como se explica en el Apartado 2.5.4.

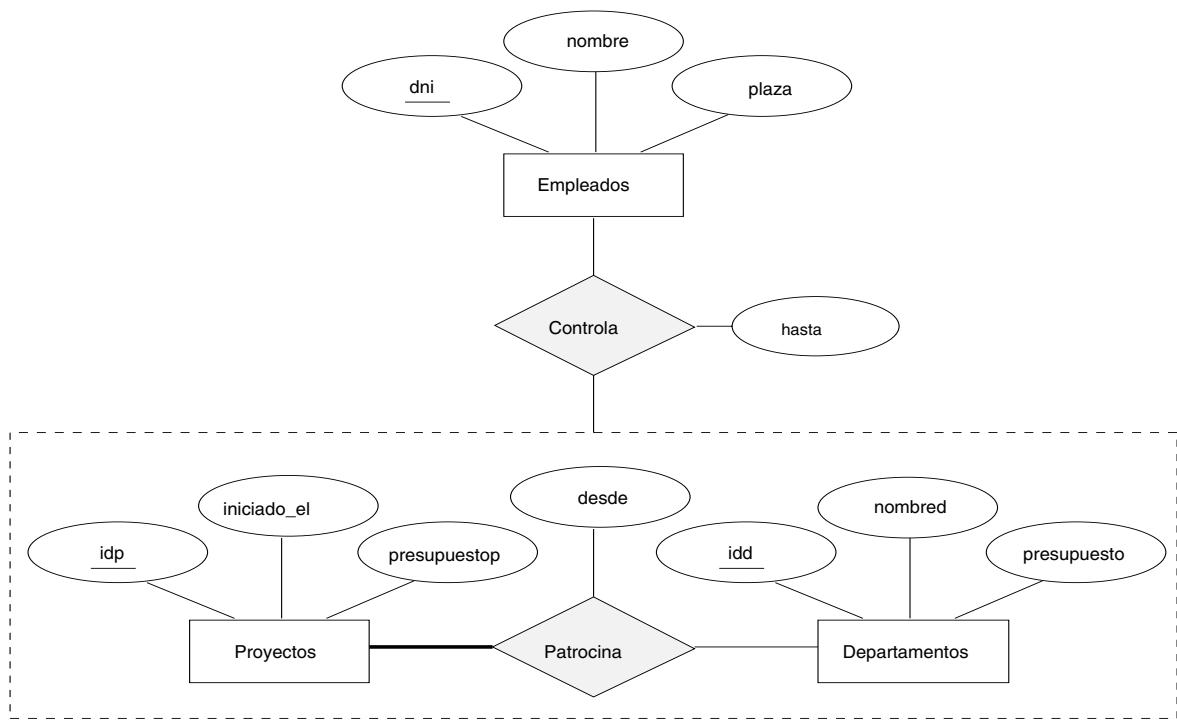


Figura 2.13 Agregación

2.5 DISEÑO CONCEPTUAL DEL MODELO ER

El desarrollo de diagramas ER supone escoger entre varias opciones, como las siguientes:

- ¿Un concepto dado se debe modelar como entidad o como atributo?
- ¿Un determinado concepto se debe modelar como entidad o como relación?
- ¿Cuáles son los conjuntos de relaciones y sus correspondientes conjuntos de entidades participantes? ¿Se deben emplear relaciones binarias o ternarias?
- ¿Se debe emplear la agregación?

Ahora se discutirán los problemas relacionados con la adopción de estas decisiones.

2.5.1 Entidades y atributos

Cuando se identifican los atributos de un conjunto de entidades no resulta a veces evidente si una determinada propiedad se debe modelar como atributo o como conjunto de entidades (y relacionarse con el primer conjunto de entidades mediante un conjunto de relaciones). Por ejemplo, considérese añadir información sobre el domicilio al conjunto de entidades Empleados. Una posibilidad es emplear el atributo *domicilio*. Esta opción resulta adecuada si sólo hace falta registrar un domicilio por empleado y basta con pensar en el domicilio como si fuera una cadena de caracteres. Una alternativa es crear un conjunto de entidades denominado Domicilios y registrar las asociaciones entre empleados y domicilios mediante una relación (por ejemplo, Tiene_domicilio). Esta alternativa más compleja resulta necesaria en dos situaciones:

- Hay que registrar más de una dirección por empleado.
- Se desea capturar la estructura de los domicilios en el diagrama ER. Por ejemplo, se puede descomponer el domicilio en ciudad, provincia, país y código postal, además de una cadena de caracteres para la información sobre la calle. Al representar el domicilio en forma de entidad con esos atributos, se pueden soportar consultas como “Buscar todos los empleados con domicilio en Madrid”.

Como ejemplo adicional de la conveniencia de modelar un concepto como conjunto de entidades en vez de como atributo, considérese el conjunto de relaciones (denominado Trabaja_en4) que puede verse en la Figura 2.14.

Sólo se diferencia del conjunto de relaciones Trabaja_en de la Figura 2.2 en que tiene los atributos *desde* y *hasta*, en lugar de sólo *desde*. De manera intuitiva, registra el intervalo durante el que cada empleado trabaja para un departamento dado. Supóngase ahora que cada empleado puede trabajar en un departamento dado en más de un periodo.

Esta posibilidad queda descartada por la semántica del diagrama ER, ya que cada relación queda identificada de maneraívoca por sus entidades participantes (recuérdese del Apartado 2.3). El problema es que se desean registrar varios valores de los atributos descriptivos de cada ejemplar de la relación Trabaja_en4. (Esta situación es análoga a desear registrar varios domicilios para cada empleado.) Se puede abordar este problema mediante la introducción

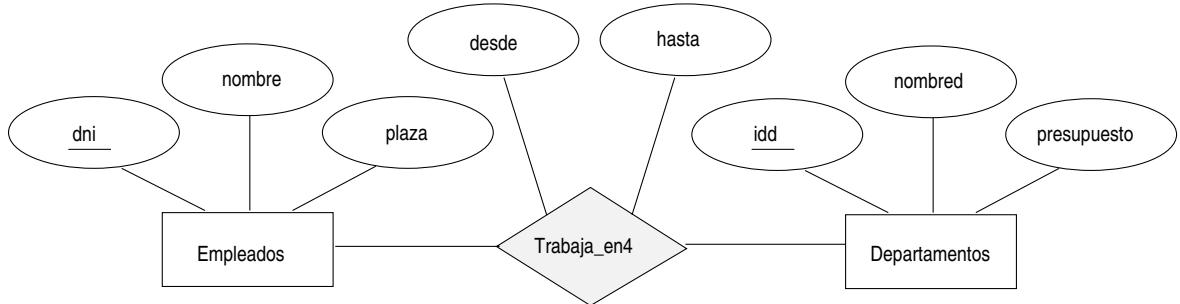


Figura 2.14 El conjunto de relaciones Trabaja_en4

de un conjunto de entidades denominado Permanencia, por ejemplo, con los atributos *desde* y *hasta*, como puede verse en la Figura 2.15.

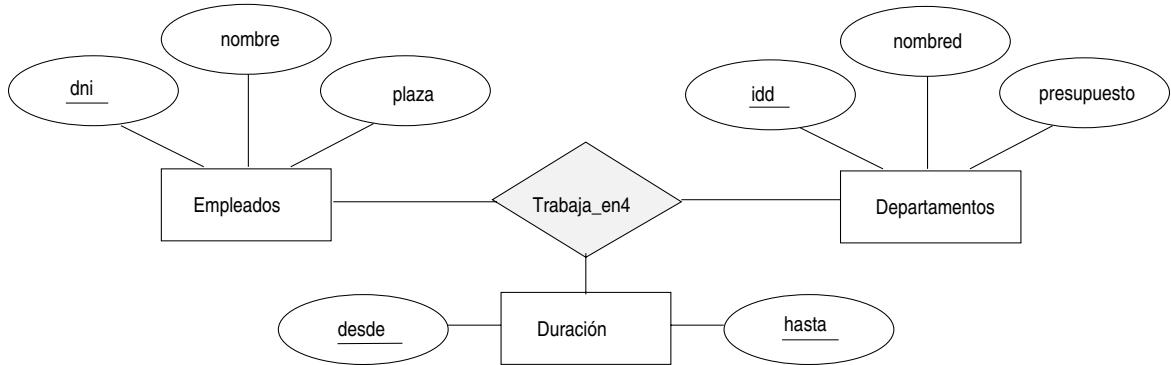
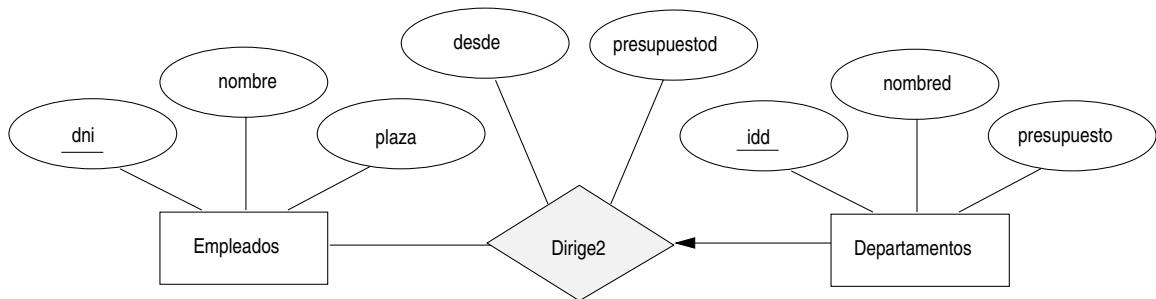


Figura 2.15 El conjunto de relaciones modificado Trabaja_en4

En algunas versiones del modelo ER se permite que los atributos adopten conjuntos como valores. Dada esta característica, se podría hacer de Permanencia un atributo de **Trabaja_en4**, en vez de un conjunto de entidades; asociado con cada relación **Trabaja_en4** tendríamos un conjunto de intervalos. Este enfoque es, quizás, más intuitivo que el modelado de Permanencia como conjunto de entidades. Pese a todo, cuando esos atributos de tipo conjunto se trasladan al modelo relacional, que no los soporta, el esquema relacional resultante es muy parecido a lo que se obtiene considerando Permanencia como conjunto de entidades.

2.5.2 Entidades y relaciones

Considérese el conjunto de relaciones denominado **Dirige** de la Figura 2.6. Supóngase que se concede a cada encargado de departamento el presupuesto discrecional (*presupuestod*), como puede verse en la Figura 2.16, en la que también se ha renombrado el conjunto de relaciones como **Dirige2**.

**Figura 2.16** Entidades y relaciones

Dado un departamento, se conoce su encargado, su fecha de nombramiento y el presupuesto de ese departamento. Este enfoque resulta natural si se da por supuesto que cada encargado recibe un presupuesto discrecional diferente para cada departamento que dirige.

¿Pero qué ocurre si el presupuesto discrecional es una suma que abarca a *todos* los departamentos dirigidos por ese empleado? En ese caso, cada relación *Dirige2* que implique a un empleado dado tendrá el mismo valor del campo *presupuestod*, lo que lleva al almacenamiento redundante de la misma información. Otro problema de este diseño es que induce a error; sugiere que el presupuesto está asociado a la relación cuando, en realidad, está asociado al encargado.

Estos problemas se pueden abordar mediante la introducción de un nuevo conjunto de entidades denominado *Encargados* (que puede ubicarse por debajo de *Empleados* en la jerarquía ES, para mostrar que todos los encargados son también empleados). Los atributos *desde* y *presupuestod* describen ahora a entidades *encargado*, como se pretendía. A modo de variación, aunque cada encargado tiene un presupuesto, cada uno de ellos puede tener una fecha de nombramiento (como encargado) diferente para cada departamento. En ese caso, *presupuestod* es un atributo de *Encargados*, pero *desde* es un atributo del conjunto de relaciones entre los encargados y los departamentos.

La naturaleza imprecisa del modelado ER puede, por tanto, dificultar el reconocimiento de las entidades subyacentes, y puede que se asocien atributos con relaciones en lugar de con las entidades correspondientes. En general, esos errores llevan al almacenamiento redundante de la misma información y pueden provocar muchos problemas. La redundancia y sus problemas asociados se discuten en el Capítulo 12, y se presenta una técnica denominada *normalización* para eliminar las redundancias de las tablas.

2.5.3 Relaciones binarias y ternarias

Considérese el diagrama ER de la Figura 2.17. Modela una situación en la que cada empleado puede tener varias pólizas, cada póliza puede ser propiedad de varios empleados y cada beneficiario puede estar cubierto por varias pólizas.

Supóngase que se tienen los requisitos adicionales siguientes:

- Dos o más empleados no pueden poseer conjuntamente una póliza.

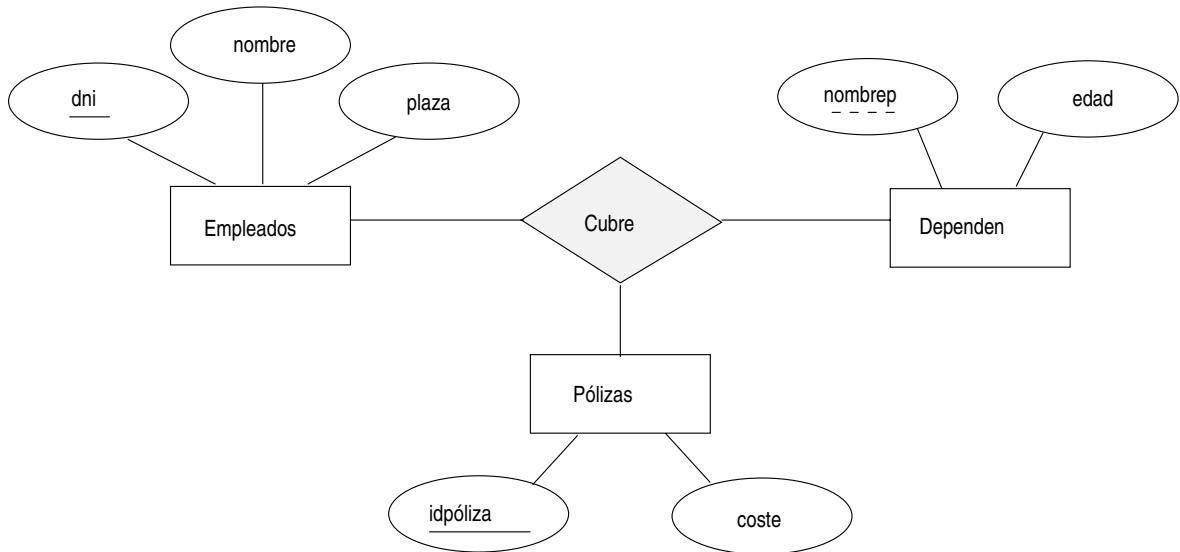


Figura 2.17 Las pólizas como conjunto de entidades

- Cada póliza debe ser propiedad de algún empleado.
- Dependientes es un conjunto de entidades débiles, y cada entidad dependiente queda identificada de manera única empleando *nombrep* junto con *idpóliza* de una entidad póliza (que, de manera intuitiva, cubre a ese beneficiario).

El primer requisito sugiere que se ha impuesto una restricción de clave a Pólizas con respecto a Cubre, pero esa restricción tiene el efecto secundario no deseado de que cada póliza sólo pueda cubrir a un beneficiario. El segundo requisito sugiere que se ha impuesto una restricción de participación total sobre Pólizas. Esta solución resulta aceptable si cada póliza cubre, por lo menos, a un beneficiario. El tercer requisito nos obliga a introducir una relación identificadora que sea binaria (en esta versión de los diagramas ER, aunque hay versiones en las que no es ése el caso).

Aun ignorando el tercer requisito, la mejor manera de modelar esta situación es emplear dos relaciones binarias, como puede verse en la Figura 2.18.

Este ejemplo tiene realmente dos relaciones que implican a Pólizas, e intentar emplear una sola relación ternaria (Figura 2.17) resulta inadecuado. Hay situaciones, no obstante, en las que una relación asocia de manera inherente a más de dos entidades. Se ha visto un ejemplo de ello en las Figuras 2.4 y 2.15.

Como ejemplo típico de relación ternaria, considérense los conjuntos de entidades Repuestos, Proveedores y Departamentos, y el conjunto de relaciones Contratos (con el atributo descriptivo *cant*) que los implica a todos ellos. Un contrato dado especifica que un determinado proveedor suministrará (una determinada cantidad de) un repuesto concreto a un cierto departamento. Esta relación no puede capturarse de manera adecuada mediante un conjunto de relaciones binarias (sin el empleo de la agregación). Con las relaciones binarias se puede denotar que un proveedor “puede suministrar” determinados repuestos, que un departamento

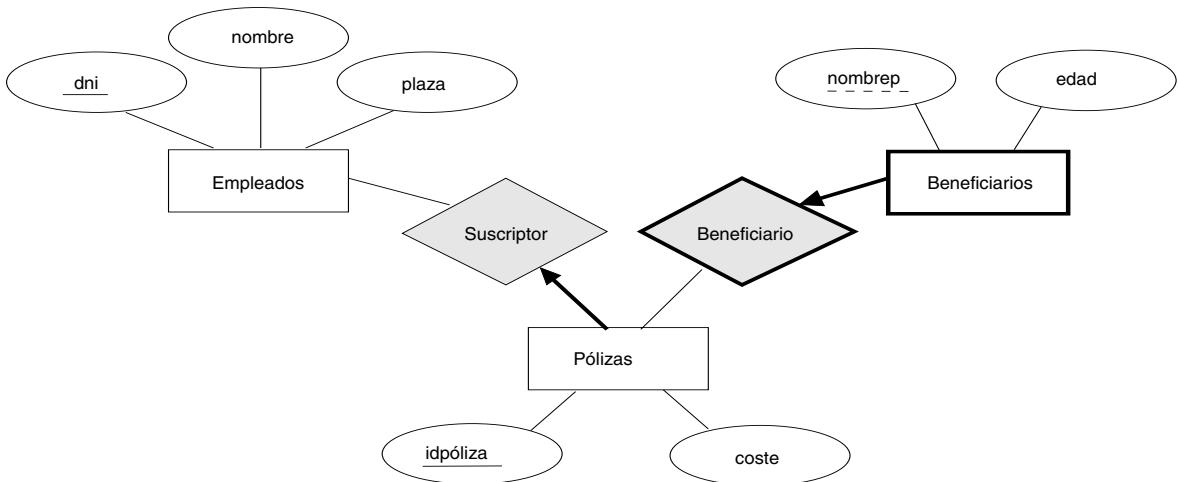


Figura 2.18 Las pólizas de nuevo

“necesita” ciertos repuestos o que un departamento “trata con” un proveedor dado. Ninguna combinación de estas relaciones expresa de manera adecuada el significado de un contrato, al menos por dos razones:

- El hecho de que el proveedor P pueda suministrar el repuesto R, de que el departamento D necesite el repuesto R y de que D compre a P no implica necesariamente que el departamento D compre realmente el repuesto R al proveedor P.
- No se puede representar adecuadamente el atributo *cant* de los contratos.

2.5.4 Agregación y relaciones ternarias

Como se señaló en el Apartado 2.4.5, la decisión de emplear la agregación o una relación ternaria viene determinada principalmente por la existencia de una relación que vincule un *conjunto de relaciones* con un conjunto de entidades (o un segundo conjunto de relaciones). Puede que la decisión también se guíe por determinadas restricciones de integridad que se deseen expresar. Por ejemplo, considérese el diagrama ER mostrado en la Figura 2.13. De acuerdo con ese diagrama, cada proyecto puede ser patrocinado por varios departamentos, cada departamento puede patrocinar uno o varios proyectos y cada patrocinio está controlado por uno o varios empleados. Si no hace falta registrar el atributo *hasta* de Controla, puede resultar razonable emplear una relación ternaria como, por ejemplo, Patrocina2, como puede verse en la Figura 2.19.

Considérese la restricción de que cada patrocinio (de un proyecto por un departamento) esté controlado, como máximo, por un empleado. No se puede expresar esa restricción en términos del conjunto de relaciones Patrocina2. Por otro lado, esa restricción se puede expresar fácilmente trazando una flecha desde la relación agregada Patrocina a la relación Controla de la Figura 2.13. Por tanto, la presencia de una restricción así es un motivo más para el empleo de la agregación en lugar de un conjunto de relaciones ternarias.

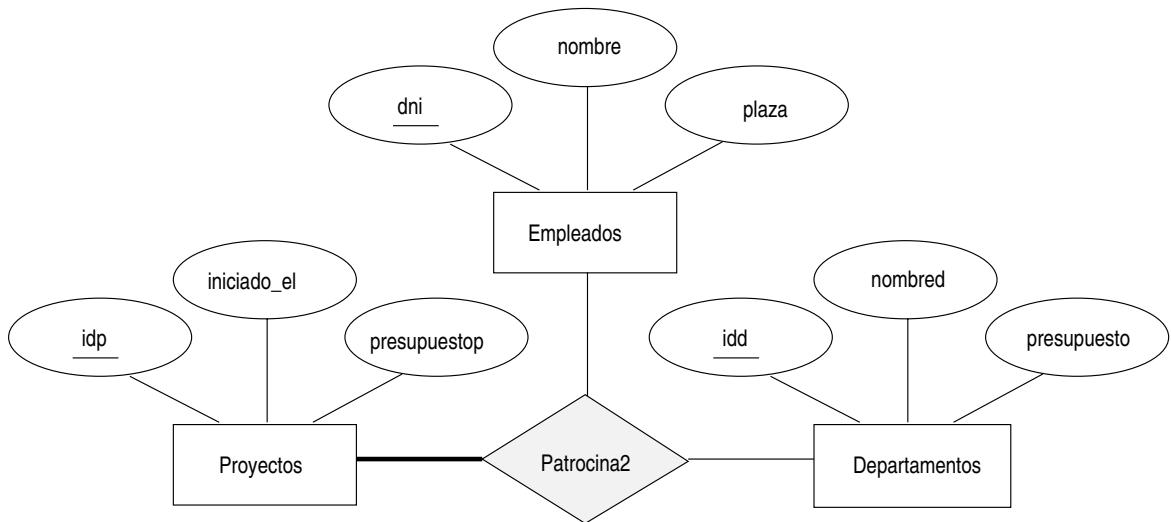


Figura 2.19 Empleo de una relación ternaria en lugar de la agregación

2.6 DISEÑO CONCEPTUAL PARA GRANDES EMPRESAS

Hasta ahora nos hemos concentrado en las estructuras disponibles en el modelo ER para la descripción de diversos conceptos y relaciones de las aplicaciones. El proceso de diseño conceptual consiste en algo más que la mera descripción de pequeños fragmentos de la aplicación en términos del diagrama ER. Para las grandes empresas puede que el diseño exija el esfuerzo de más de un diseñador y abarque datos y código de aplicaciones empleado por varios grupos de usuarios. El empleo de un modelo de datos semántico de alto nivel, como los diagramas ER, para el diseño conceptual en un entorno de este tipo ofrece la ventaja adicional de que el diseño de alto nivel se puede representar mediante diagramas y lo puede comprender fácilmente la gran cantidad de personas que deben ofrecer aportaciones al proceso de diseño.

Un aspecto importante del proceso de diseño es la metodología empleada para estructurar el desarrollo del diseño global y garantizar que tenga en cuenta todos los requisitos de usuario y sea consistente. El enfoque habitual es que se tomen en consideración los requisitos de diversos grupos de usuarios, se resuelvan de algún modo los que entren en conflicto y se genere un conjunto único de requisitos globales al final de la fase de análisis de requisitos. La generación de un conjunto único de requisitos globales es una labor difícil, pero permite que la fase de diseño conceptual se continúe con el desarrollo de un esquema lógico que abarque todos los datos y a todas las aplicaciones de la empresa.

Un enfoque alternativo es el desarrollo de esquemas conceptuales independientes para los diferentes grupos de usuarios y su posterior *integración*. Para integrar varios esquemas conceptuales hay que establecer correspondencias entre las entidades, las relaciones y los atributos, y hay que resolver numerosos tipos de conflictos (por ejemplo, de denominaciones, incoherencias de dominios, diferencias en las unidades de medida, etcétera). Esta tarea es

intrínsecamente difícil. En algunas situaciones la integración de los esquemas no se puede evitar; por ejemplo, cuando una organización se fusiona con otra, puede que haya que integrar las bases de datos existentes. La integración de esquemas también aumenta de importancia a medida que los usuarios van exigiendo el acceso a orígenes de datos *heterogéneos*, a menudo mantenidos por organizaciones diferentes.

2.7 EL LENGUAJE UNIFICADO DE MODELADO

Hay muchos enfoques del diseño integral de software, que abarcan todas las etapas desde la identificación de los requisitos empresariales a las especificaciones finales para una aplicación completa, incluidos el flujo de trabajo, las interfaces de usuario y muchos aspectos de los sistemas de software que superan ampliamente las bases de datos y los datos que se guardan en ellas. En este apartado se tratará brevemente un enfoque que está ganando popularidad, denominado **enfoque del lenguaje unificado de modelado** (unified modeling language, **UML**).

UML, al igual que el modelo ER, presenta la atractiva característica de que sus estructuras se pueden dibujar en forma de diagrama. Abarca un espectro del proceso de diseño de software más amplio que el modelo ER:

- **Modelado de la empresa.** En esta fase el objetivo es describir los procesos empresariales implicados en la aplicación de software que se está desarrollando.
- **Modelado del sistema.** La comprensión de los procesos empresariales se emplea para identificar los requisitos de la aplicación de software. Una parte de esos requisitos son los de la base de datos.
- **Modelado conceptual de la base de datos.** Esta etapa corresponde a la creación del diseño ER para la base de datos. Con ese objetivo, UML ofrece muchas estructuras análogas a las del modelo ER.
- **Modelado físico de la base de datos.** UML ofrece también representaciones gráficas de las opciones de diseño físico de la base de datos, como la creación de espacios de tablas y de índices. (El diseño físico de la base de datos se discute en capítulos posteriores, pero no así las estructuras de UML correspondientes.)
- **Modelado del sistema de hardware.** Los diagramas UML se pueden emplear para describir la configuración de hardware empleada para la aplicación.

Hay muchas clases de diagramas UML. Los diagramas de **casos de uso** describen las acciones llevadas a cabo por el sistema en respuesta a las solicitudes de los usuarios, y las personas implicadas en esas acciones. Estos diagramas especifican la funcionalidad externa que se espera que soporte el sistema.

Los diagramas de **actividad** muestran el flujo de acciones en los procesos comerciales. Los diagramas de **estado** describen las interacciones dinámicas entre los diferentes objetos del sistema. Estos diagramas, empleados en el modelado de la empresa y en el del sistema, describen el modo en que se va a implementar la funcionalidad externa, de manera consistente con las reglas del negocio y con los procesos de la empresa.

Los diagramas de **clases** son parecidos a los diagramas ER, aunque son más generales en el sentido de que se pretende que modelen entidades de *aplicación* (de manera intuitiva, componentes importantes del programa) y sus relaciones lógicas, además de las entidades de datos y sus relaciones.

Tanto los conjuntos de entidades como los de relaciones pueden representarse en UML como clases, junto con las restricciones de las claves, las entidades débiles y las jerarquías de clase. El término *relación* se emplea en UML de manera ligeramente diferente, y las relaciones de UML son binarias. Esto conduce a veces a confusión sobre si los conjuntos de relaciones de los diagramas ER que implican a tres o más conjuntos de entidades se pueden representar directamente en UML. La confusión desaparece una vez que se comprende que todos los conjuntos de relaciones (en el sentido de ER) se representan en UML como clases; las “relaciones” binarias de UML son, en esencia, precisamente los enlaces mostrados en los diagramas ER entre los conjuntos de entidades y los de relaciones.

Los conjuntos de relaciones con restricciones de las claves se suelen omitir de los diagramas UML, y la relación se indica enlazando directamente los conjuntos de entidades implicados. Por ejemplo, considérese la Figura 2.6. La representación UML de ese diagrama ER tendría una clase para Empleados y otra para Departamentos, y la relación Dirige se mostraría mediante un enlace entre ambas clases. El enlace se puede etiquetar con un nombre y la correspondiente información de cardinalidad para mostrar que cada departamento sólo puede tener un encargado.

Como se verá en el Capítulo 3, los diagramas ER se traducen en el modelo relacional mediante la transformación de cada conjunto de entidades o de relaciones en una tabla. Además, como se verá en el Apartado 3.5.3, las tablas correspondientes a los conjuntos de relaciones de una a varias suelen omitirse mediante la inclusión de alguna información adicional sobre cada una de esas relaciones en la tabla correspondiente a alguno de los conjuntos de entidades implicados en las mismas. Por tanto, los diagramas de clases en UML se corresponden estrechamente con las tablas creadas mediante la transformación de los diagramas ER.

En realidad, cada clase de los diagramas UML se transforma en una tabla del diagrama UML de la base de datos correspondiente. Los **diagramas de bases de datos** en UML muestran el modo en que se representan las clases en la base de datos y contienen detalles adicionales sobre la estructura de la base de datos, como las restricciones de integridad y los índices. Los enlaces (“relaciones” de UML) entre las clases de UML conducen a diversas restricciones de integridad entre las tablas correspondientes. Muchos detalles específicos del modelo relacional (por ejemplo, las *vistas*, las *claves externas*, los *campos que pueden tener valores nulos*) y que reflejan opciones del diseño físico (por ejemplo, los campos indexados) se pueden modelar en los diagramas de bases de datos en UML.

Los diagramas de **componentes** en UML describen los aspectos de almacenamiento de las bases de datos, como los *espacios de tablas* y las *particiones de las bases de datos*, así como las interfaces con las aplicaciones que tienen acceso a la base de datos. Finalmente, los diagramas de **implantación** muestran los aspectos hardware del sistema.

El objetivo de este libro es concentrarse en los datos almacenados en las bases de datos y en los aspectos de diseño relacionados. Para ello se adoptará una visión simplificada de las otras etapas del diseño y desarrollo del software. Más allá de la discusión concreta de UML, se pretende que el material de este apartado sitúe los aspectos de diseño que se tratan en el contexto más amplio del diseño de software. Los autores esperan que esto ayude a los lectores

interesados en una discusión más general del diseño de software a complementar este estudio mediante referencias a otros materiales de su enfoque preferido al diseño global de sistemas.

2.8 ESTUDIO DE UN CASO: LA TIENDA EN INTERNET

Como ilustración se introducirá ahora el estudio de un caso de diseño desde su comienzo hasta su fin que se empleará como ejemplo permanente a lo largo del libro. TiposBD S.A., una conocida empresa de consultoría de bases de datos, ha sido requerida para ayudar a Benito y Norberto (B&N) con el diseño y la implementación de su base de datos. B&N es una gran librería especializada en los libros sobre carreras de caballos, y ha decidido abrirse a Internet. TiposBD comprueba primero que B&N desee y pueda pagar sus elevados honorarios y luego convoca una comida de trabajo —que, naturalmente, se facturará a B&N— para realizar el análisis de requisitos.

2.8.1 Análisis de requisitos

El propietario de B&N, a diferencia de muchas de las personas que necesitan una base de datos, ha meditado ampliamente lo que desea y ofrece un resumen conciso:

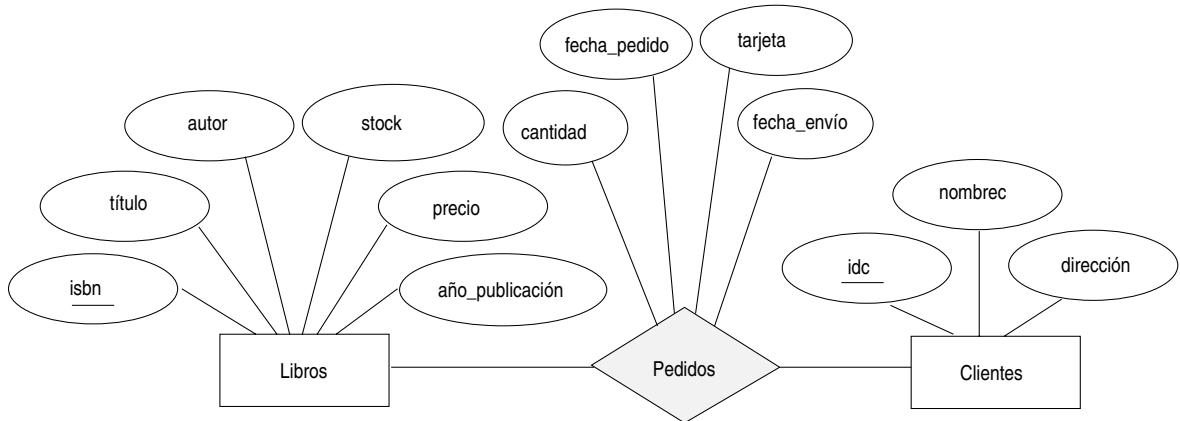
“Me gustaría que los clientes pudieran examinar el catálogo de libros y realizar pedidos por Internet. Actualmente acepto pedidos por teléfono. Tengo, sobre todo, clientes corporativos que me llaman y me dan el código ISBN del libro y la cantidad que desean comprar; a menudo pagan con tarjeta de crédito. Luego preparo el envío, que contiene los libros pedidos. Si no dispongo de suficientes copias en el almacén, encargo copias adicionales y retraso el envío hasta que llegan; prefiero enviar todo el pedido de cada cliente de una sola vez. Mi catálogo incluye todos los libros que vendo. Para cada libro, el catálogo incluye su código ISBN, título, autor, precio de adquisición, precio de venta y año de publicación. La mayor parte de mis clientes son habituales, y dispongo de un registro con su nombre y dirección. Los clientes nuevos tienen que llamarme primero y abrir una cuenta antes de poder utilizar mi Web.

En la nueva Web los clientes se deberían identificar antes de nada por su número de identificación de cliente, que debe ser único. Luego deberían poder examinar el catálogo y formular pedidos en línea.”

Los consultores de TiposBD están un poco sorprendidos por la rapidez con que se ha completado la fase de requisitos —suelen hacer falta semanas de discusiones (y muchas comidas y muchas cenas) llevarla a buen puerto— pero vuelven a su oficina para analizar esta información.

2.8.2 Diseño conceptual

En la etapa de diseño conceptual, TiposBD desarrolla una descripción de alto nivel de los datos en términos del modelo ER. El diseño inicial se puede ver en la Figura 2.20. Los libros y los clientes se modelan como entidades y se relacionan mediante los pedidos que formulan los clientes. Pide es un conjunto de relaciones que conecta los conjuntos de entidades Libros y Clientes. Para cada pedido se almacenan los atributos siguientes: cantidad, fecha del pedido

**Figura 2.20** Diagrama ER del diseño inicial

y fecha de envío. En cuanto se envía un pedido, se establece la fecha de envío; hasta entonces, la fecha de envío se define como *nula*, lo que indica que ese pedido no se ha enviado todavía.

TiposBD tiene en este momento una reunión interna para la revisión del diseño, y se suscitan varias dudas. Para proteger sus identidades, nos referiremos al jefe del equipo de diseño como Tipo 1 y al revisor del diseño como Tipo 2.

Tipo 2: ¿Qué ocurre si un cliente realiza dos pedidos del mismo libro el mismo día?

Tipo 1: Se trata el primer pedido mediante la creación de una relación Pide y el segundo mediante la actualización del valor del atributo cantidad de esa relación.

Tipo 2: ¿Qué ocurre si un cliente realiza dos pedidos para libros diferentes el mismo día?

Tipo 1: No hay problema. Cada ejemplar del conjunto de relaciones Pide relaciona al cliente con un libro diferente.

Tipo 2: Ah, ¿pero qué ocurre si un cliente realiza dos pedidos del mismo libro en días diferentes?

Tipo 1: Se puede emplear el atributo fecha de pedido de la relación Pide para distinguir los dos pedidos.

Tipo 2: Oh, no, no se puede. Los atributos de Clientes y de Libros deben contener una clave de Pide conjuntamente. Por tanto, este diseño no permite que un cliente realice pedidos del mismo libro en días diferentes.

Tipo 1: Pues vaya, tienes razón. Oh, bueno, probablemente no le importe a B&N; ya veremos.

TiposBd decide pasar a la fase siguiente, el diseño lógico de la base de datos; volveremos a verlos en el Apartado 3.8.

2.9 PREGUNTAS DE REPASO

Las respuestas a las preguntas de repaso pueden encontrarse en los apartados indicados.

- Indíquense las principales etapas del diseño de bases de datos. ¿Cuál es el objetivo de cada etapa? ¿En qué etapa se emplea principalmente el modelo ER? (**Apartado 2.1**)
- Defínanse estos términos: *entidad*, *conjunto de entidades*, *atributo*, *clave*. (**Apartado 2.2**)

- Defínanse estos términos: *relación, conjunto de relaciones, atributos descriptivos*. (**Apartado 2.3**)
- Defínanse los siguientes tipos de restricciones y dese un ejemplo de cada una: *restricción de la clave, restricción de participación*. ¿Qué es una *entidad débil*? ¿Qué son las *jerarquías de clases*? ¿Qué es la *agregación*? Dese una situación de ejemplo que motive el empleo de cada una de estas estructuras de diseño del modelo ER. (**Apartado 2.4**)
- ¿Qué directrices se emplearían para cada una de estas opciones en la realización de un diseño ER? emplear un atributo o un conjunto de entidades, una entidad o un conjunto de relaciones, una relación binaria o una ternaria, o la agregación. (**Apartado 2.5**)
- ¿Por qué resulta especialmente difícil diseñar una base de datos para una gran empresa? (**Apartado 2.6**)
- ¿Qué es UML? ¿Cómo encaja el diseño de bases de datos en el diseño global de un sistema de software con empleo intensivo de los datos? ¿Cómo se relaciona UML con los diagramas ER? (**Apartado 2.7**)

EJERCICIOS

Ejercicio 2.1 Explíquense brevemente los términos siguientes: *atributo, dominio, entidad, relación, conjunto de entidades, conjunto de relaciones, relación de una a varias, relación de varias a varias, restricción de participación, restricción de solapamiento, restricción de cobertura, conjunto de entidades débiles, agregación e indicador de papel*.

Ejercicio 2.2 La base de datos de una universidad contiene información sobre los profesores (identificados por su documento nacional de identidad, o DNI) y las asignaturas (identificadas por idasignatura). Los profesores imparten las asignaturas; cada una de las situaciones siguientes concierne al conjunto de relaciones Imparte. Para cada situación, dibújese un diagrama que la describa (suponiendo que no se aplica ninguna otra restricción).

1. Los profesores pueden impartir la misma asignatura en varios semestres, y cada ocasión debe registrarse.
2. Los profesores pueden impartir el mismo curso en varios semestres, y sólo hace falta registrar la ocasión más reciente. (Supóngase que esta condición es aplicable a las preguntas siguientes.)
3. Cada profesor debe impartir alguna asignatura.
4. Cada profesor imparte exactamente una asignatura (ni más ni menos).
5. Cada profesor imparte exactamente una asignatura (ni más ni menos), y cada asignatura debe impartirla algún profesor.
6. Supóngase ahora que determinadas asignaturas puede impartirlas conjuntamente un equipo de profesores, pero que es posible que ningún profesor de ese equipo pueda impartir esa asignatura. Modélese esta situación, introduciendo más conjuntos de entidades y de relaciones si fuera necesario.

Ejercicio 2.3 Considérese la siguiente información sobre la base de datos de una universidad:

- Cada profesor tiene DNI, nombre, edad, rango y especialidad de investigación.
- Cada proyecto tiene número de proyecto, nombre de patrocinador (por ejemplo, el CSIC), fecha de comienzo, fecha de finalización y presupuesto.
- Cada alumno de posgrado tiene DNI, nombre, edad y programa de posgrado (por ejemplo, magíster o doctorado).

52 Sistemas de gestión de bases de datos

- Cada proyecto está dirigido por un profesor (conocido como investigador principal de ese proyecto).
- En cada proyecto trabajan uno o varios profesores (conocidos como investigadores de ese proyecto).
- Cada profesor puede dirigir varios proyectos o trabajar en ellos.
- En cada proyecto trabajan uno o varios alumnos de posgrado (conocidos como ayudantes de investigación de ese proyecto).
- Cuando los alumnos de posgrado trabajan en un proyecto, su trabajo debe supervisarlo un profesor. Cada alumno de posgrado puede trabajar en varios proyectos, en cuyo caso puede tener un supervisor diferente en cada uno de ellos.
- Cada departamento tiene número de departamento, nombre de departamento y despacho principal.
- Cada departamento tiene un profesor (conocido como director) que lo dirige.
- Cada profesor trabaja en uno o varios departamentos, y por cada departamento en que trabaja se asocia un porcentaje de tiempo a su trabajo.
- Cada alumno de posgrado tiene un departamento principal en el que trabaja en su titulación.
- Cada alumno de posgrado tiene otro alumno de posgrado más veterano (conocido como asesor del alumno) que lo aconseja sobre las asignaturas en las que debe matricularse.

Diséñese y dibújese un diagrama ER que capture la información sobre la universidad. Empléense únicamente el modelo ER básico, es decir, las entidades, las relaciones y los atributos. Asegúrese de indicar las claves y las restricciones de participación que pueda haber.

Ejercicio 2.4 La base de datos de una empresa necesita almacenar información sobre los empleados (identificados por su *dni*, con *sueldo* y *teléfono* como atributos); los departamentos (identificados por *númd*, con *nombred* y *presupuesto* como atributos); y los hijos de los empleados (con *nombre* y *edad* como atributos). Los empleados *trabajan* en los departamentos; cada departamento está *dirigido por* un empleado; cada hijo debe quedar identificado de manera única por su *nombre* una vez conocidos su padre o su madre (que es uno de los empleados; supóngase que sólo uno de los padres trabaja en la empresa). No se está interesado en la información relativa al hijo una vez que el padre deja la empresa.

Dibújese un diagrama ER que capture esta información.

Ejercicio 2.5 Discos Sinpueblo (Notown Records) ha decidido guardar información sobre los músicos que intervienen en sus discos (así como otros datos de la empresa) en una base de datos. Sabiamente, la empresa ha decidido contratarlo a usted como diseñador de la base de datos (por su tarifa habitual de 2500 €diarios).

- Cada músico que graba en Sinpueblo tiene DNI, nombre, dirección y número de teléfono. Los músicos que cobran poco suelen compartir la misma dirección, y ninguna dirección tiene más de un teléfono.
- Cada instrumento que se emplea en las canciones grabadas en Sinpueblo tiene nombre (por ejemplo, guitarra, sintetizador o flauta) y una clave musical (por ejemplo, re, si bemol, mi bemol).
- Cada disco que se graba en el sello Sinpueblo tiene título, fecha de copyright, formato (por ejemplo, CD o MC) e identificador del disco.
- Cada canción grabada en Sinpueblo tiene título y autor.
- Cada músico puede tocar varios instrumentos, y cada instrumento pueden tocarlo varios músicos.
- Cada disco tiene varias canciones, pero ninguna canción puede aparecer en más de un disco.
- Cada canción la interpretan uno o varios músicos, y cada músico puede interpretar varias canciones.
- Cada disco tiene exactamente un músico que actúa como productor. Por supuesto, cada músico puede producir varios discos.

Diséñese un esquema conceptual para Sinpueblo y dibújese un diagrama ER para ese esquema. La información siguiente describe la situación que debe modelar la base de datos de Sinpueblo. Asegúrese de indicar todas las restricciones de clave y cardinalidad, y cualquier suposición que se haga. Identifíquense todas las restricciones que no se puedan capturar en el diagrama ER y explíquese brevemente el motivo de no poder expresarlas.

Ejercicio 2.6 Los miembros del Departamento de Informática que vuelan a menudo se han quejado al personal del Aeropuerto de Daganzo de la mala organización de ese aeropuerto. En consecuencia, el personal del aeropuerto ha decidido que toda la información relativa al aeropuerto se organice mediante un SGBD, y se le ha contratado para diseñar la correspondiente base de datos. Su primera tarea es organizar la información relativa a los aviones que se albergan y realizan su mantenimiento en el aeropuerto. La información relevante es la siguiente:

- Cada avión tiene un número de registro y es de un modelo concreto.
 - El aeropuerto puede atender a varios modelos de aeroplano, y cada modelo está identificado por un número de modelo (por ejemplo, DC-10) y tiene una capacidad y un peso dados.
 - En el aeropuerto trabajan varios técnicos. Hay que guardar el nombre, DNI, dirección, número de teléfono y sueldo de cada uno de ellos.
 - Cada técnico es experto en uno o varios modelos de avión, y su maestría puede solaparse con la de otros técnicos. También hay que guardar esta información sobre los técnicos.
 - Los controladores aéreos deben pasar un examen médico anual. Para cada controlador aéreo hay que guardar la fecha del examen más reciente.
 - Todos los empleados del aeropuerto (incluidos los técnicos) pertenecen a un sindicato. Hay que guardar el número de afiliación al sindicato de cada empleado. Se puede suponer que cada empleado queda identificado de manera unívoca por el número de su documento nacional de identidad.
 - El aeropuerto tiene varias pruebas que se emplean periódicamente para garantizar que los aviones siguen estando en condiciones de volar. Cada prueba tiene un número de examen de la Dirección General de Aviación Civil (DGAC), un nombre y una puntuación máxima posible.
 - La DGAC exige que el aeropuerto registre el momento en que cada avión es examinado por un técnico dado siguiendo una prueba concreta. Para cada examen la información necesaria es la fecha, el número de horas que el técnico ha empleado en realizarlo y la puntuación que el avión ha conseguido.
1. Dibújese un diagrama ER para la base de datos del aeropuerto. Asegúrese de indicar los diferentes atributos de cada entidad y de cada conjunto de relaciones; especifíquense también las restricciones de clave y participación de cada conjunto de relaciones. Especifíquense las restricciones de solapamiento y de cobertura necesarias (en su lengua materna).
 2. La DGAC ha aprobado una norma que obliga a que los exámenes de los aviones los realicen técnicos expertos en cada modelo. ¿Cómo se expresaría esa restricción en el diagrama ER? Si no se puede expresar, explíquese el motivo brevemente.

Ejercicio 2.7 La cadena de farmacias Recetas Rayos X le ha ofrecido abastecerlo gratuitamente de medicamentos de por vida si diseña su base de datos. Dado el creciente coste de la atención sanitaria, usted acepta. Esta es la información que logra reunir:

- Los pacientes se identifican mediante su DNI y hay que registrar su nombre, dirección y edad.
- Los médicos se identifican mediante su DNI. Para cada médico hay que registrar el nombre, la especialidad y los años de ejercicio.
- Cada empresa farmacéutica se identifica por el nombre y tiene un número de teléfono.
- Para cada medicamento hay que registrar el nombre comercial y la fórmula. Cada medicamento lo vende una empresa farmacéutica dada, y el nombre comercial identifica ese medicamento de manera unívoca entre los productos de esa empresa. Si se borra una empresa farmacéutica, ya no hace falta realizar el seguimiento de sus productos.
- Cada farmacia tiene nombre, dirección y número de teléfono.
- Cada paciente tiene un médico de cabecera. Cada médico tiene, como mínimo, un paciente.
- Cada farmacia vende varios medicamentos y tiene un precio para cada uno de ellos. Cada medicamento se puede vender en varias farmacias, y el precio puede variar de una a otra.

54 Sistemas de gestión de bases de datos

- Los médicos recetan drogas a sus pacientes. Cada médico puede recetar una o varias drogas a varios pacientes, y cada paciente puede conseguir recetas de varios médicos. Cada receta tiene una fecha y una cantidad asociadas con ella. Se puede suponer que, si un médico receta el mismo medicamento al mismo paciente más de una vez, sólo hay que guardar la última receta.
- Las empresas farmacéuticas tienen contratos de larga duración con las farmacias. Cada empresa farmacéutica puede contratar con varias farmacias, y cada farmacia puede contratar con varias empresas farmacéuticas. Para cada contrato hay que guardar la fecha de inicio, la fecha de finalización y el texto íntegro.
- Las farmacias nombran un supervisor para cada contrato. Siempre debe haber un supervisor por contrato, pero el supervisor de un contrato puede cambiar durante su periodo de validez.
 1. Dibújese un diagrama ER que capture esta información. Identifíquense las restricciones que no queden capturadas por el diagrama ER.
 2. ¿Cómo cambiaría el diseño si cada medicamento debiera venderlo todas las farmacias al mismo precio?
 3. ¿Cómo cambiaría el diseño si los requisitos de diseño cambiaran de la manera siguiente? Si un médico receta el mismo medicamento al mismo paciente más de una vez, puede que haya que guardar varias de esas recetas.

Ejercicio 2.8 Aunque siempre ha deseado ser artista, acabó siendo experto en bases de datos porque le encantan los animales y, de alguna manera, confundió “base de datos” con “base de gatos”. Ese antiguo amor sigue ahí, sin embargo, por lo que ha montado una empresa de bases de datos, BaseArt, que crea productos para galerías de arte. El corazón del producto es una base de datos con un esquema que captura toda la información que necesitan guardar las galerías. Las galerías guardan información de los artistas: nombre (que es único), lugar de nacimiento, edad y estilo artístico. Para cada pieza de arte hay que guardar el autor, el año de realización, el título (que es único), la rama artística (por ejemplo, pintura, litografía, escultura o fotografía) y el precio. Las piezas de arte también se clasifican en grupos de diferentes tipos como, por ejemplo, retratos, bodegones, obras de Picasso u obras del siglo diecinueve; cada pieza puede pertenecer a más de un grupo. Cada grupo se identifica mediante un nombre (como los anteriores) que lo describe. Finalmente, las galerías guardan información sobre sus clientes. Para cada cliente las galerías guardan el nombre (que es único), la dirección, el total de euros que se ha gastado en esa galería (*¡muy importante!*) y los artistas y grupos de arte que más le gustan.

Dibújese el diagrama ER de esta base de datos.

Ejercicio 2.9 Respóndanse las siguientes preguntas.

- Explíquese brevemente los siguientes términos: *UML, diagramas de casos de uso, diagramas de estado, diagramas de clases, diagramas de bases de datos, diagramas de componentes y diagramas de implantación.*
- Explíquese la relación entre los diagramas ER y UML.

NOTAS BIBLIOGRÁFICAS

Hay varios libros que ofrecen un buen tratamiento del diseño conceptual; entre ellos están [43] (que también contiene un estudio de las herramientas comerciales para el diseño de bases de datos) y [442].

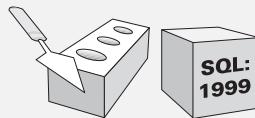
El modelo ER fue propuesto por Chen [107] y se han propuesto ampliaciones del mismo en varios trabajos posteriores. La generalización y la agregación se introdujeron en [417]. [254] y [354] contienen buenos estudios sobre los modelos semánticos de datos. Los aspectos dinámicos y temporales de los modelos semánticos de datos se tratan en [457].

[443] estudia una metodología de diseño basada en el desarrollo del diagrama ER y su posterior traducción al modelo relacional. Markowitz considera la integridad referencial en el contexto de la transformación del

modelo ER al relacional y estudia el soporte ofrecido en varios sistemas comerciales (hasta la fecha de su publicación) en [318, 319].

Las actas de la conferencia entidad-relación contienen numerosos trabajos sobre diseño conceptual, con cierto énfasis en el modelo ER como, por ejemplo, [421].

La integración de las vistas se discute en varios trabajos, como [57, 85, 116, 156, 334, 338, 337, 411, 420, 456]. [44] es un estudio de varios enfoques a la integración.



3

EL MODELO RELACIONAL

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

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

— *Diccionario Webster de la lengua inglesa*

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

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

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

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

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

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

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

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

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

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

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

3.1 INTRODUCCIÓN AL MODELO RELACIONAL

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

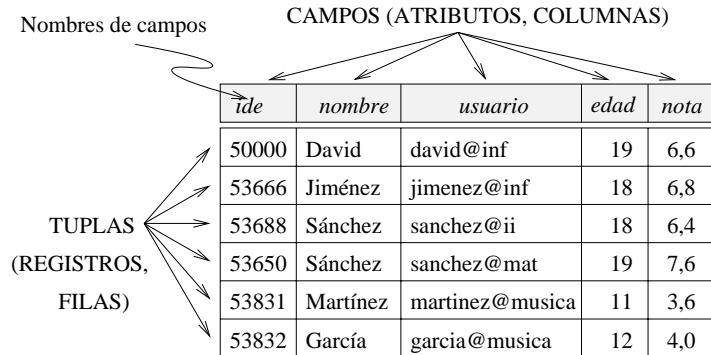
Para ilustrar las partes del esquema de una relación se empleará el ejemplo introducido en el Capítulo 1 sobre la información de alumnos en la base de datos de una universidad:

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

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

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

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

**Figura 3.1** El ejemplar A1 de la relación Alumnos

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

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

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

Figura 3.2 Una representación alternativa del ejemplar A1 de Alumnos

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

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

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

De manera más formal, sea $R(c_1:D_1, \dots, c_n:D_n)$ el esquema de una relación y, para cada c_i , $1 \leq i \leq n$, sea Dom_i el conjunto de valores asociados con el dominio denominado D_i . Cada ejemplar de R que cumple las restricciones de dominio del esquema es un conjunto de tuplas con n campos:

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

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

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

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

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

3.1.1 Creación y modificación de relaciones mediante SQL

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

62 Sistemas de gestión de bases de datos

La instrucción `CREATE TABLE` se emplea para definir tablas nuevas¹. Para crear la relación Alumnos se puede emplear la instrucción siguiente:

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

Las tuplas se insertan mediante la orden `INSERT`. Se puede insertar una sola tupla en la tabla Alumnos de la manera siguiente²:

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

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

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

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

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

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

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

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

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

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

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

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

Figura 3.3 El ejemplar A1 de Alumnos tras la actualización

3.2 RESTRICCIONES DE INTEGRIDAD SOBRE RELACIONES

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

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

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

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

3.2.1 Restricciones de clave

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

Examinemos más de cerca la definición anterior de clave (candidata). La definición tiene dos partes³:

1. Dos tuplas distintas de un ejemplar legal (un ejemplar que cumple todas las RI, incluida la restricción de clave) no pueden tener valores idénticos en todos los campos de una clave.
2. Ningún subconjunto del conjunto de campos de una clave es identificador único de una tupla.

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

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

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

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

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

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

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

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

Especificación de restricciones de clave en SQL

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

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

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

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

3.2.2 Restricciones de clave externa

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

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

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

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

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

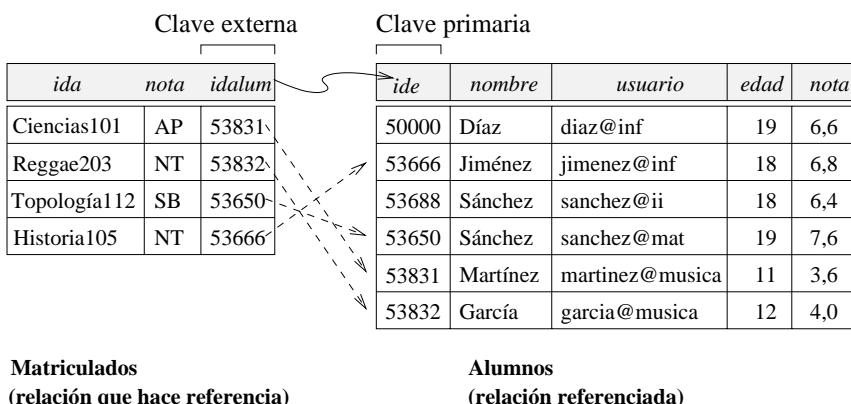


Figura 3.4 Integridad referencial

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

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

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

Especificación de restricciones de clave externa en SQL

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

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

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

3.2.3 Restricciones generales

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

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

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

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

Figura 3.5 El ejemplar A2 de la relación Alumnos

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

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

3.3 CUMPLIMIENTO DE LAS RESTRICCIONES DE INTEGRIDAD

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

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

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

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

```
INSERT
```

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

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

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

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

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

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

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

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

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

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

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

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

Las opciones son:

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

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

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

Las opciones son parecidas a las del caso anterior.

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

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

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

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

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

3.3.1 Transacciones y restricciones

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

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

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

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

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

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

SET CONSTRAINT Restricción DEFERRED

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

3.4 CONSULTAS DE DATOS RELACIONALES

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

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

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

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

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

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

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

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

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

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

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

nombre	usuario
Martínez	martinez@musica
García	garcia@musica

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

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

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

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

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

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

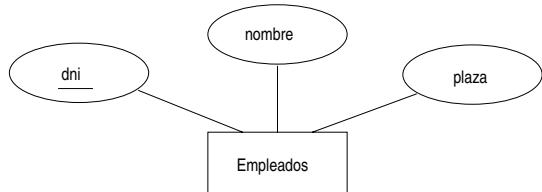
El modelo ER resulta conveniente para representar un diseño inicial de alto nivel de la base de datos. Dado un diagrama ER que describe una base de datos, se sigue el enfoque estándar para generar un esquema relacional de la base de datos que se aproxime lo más posible al diseño ER. (La traducción es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

3.5.1 De los conjuntos de entidades a las tablas

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

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

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

**Figura 3.8** El conjunto de entidades Empleados

<i>dni</i>	<i>nombre</i>	<i>plaza</i>
123.223.666	Avelino	48
231.315.368	Serna	22
131.243.650	Soria	35

Figura 3.9 Un ejemplar del conjunto de entidades Empleados

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

```

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

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

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

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

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

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

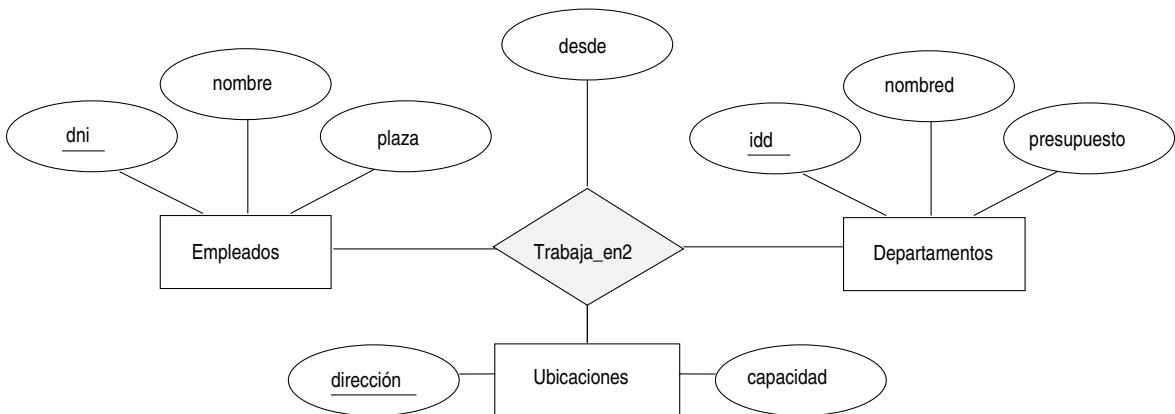


Figura 3.10 Un conjunto de relaciones ternarias

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

```

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

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

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

```

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

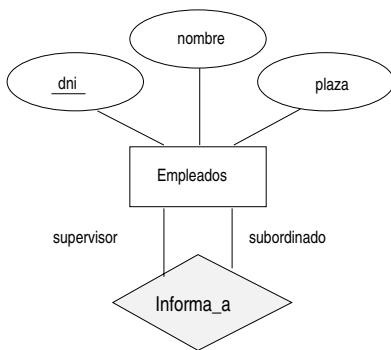


Figura 3.11 El conjunto de relaciones Inform_a

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

3.5.3 Traducción de conjuntos de relaciones con restricciones de clave

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

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

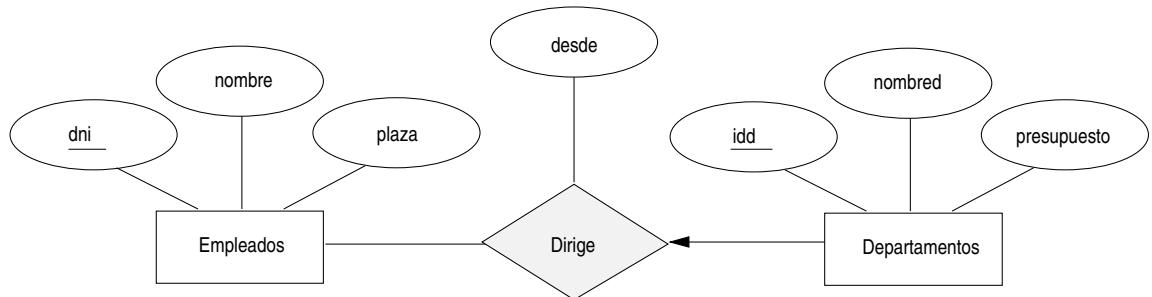


Figura 3.12 Restricción de clave para Dirige

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

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

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

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

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

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

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

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

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

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

3.5.4 Traducción de los conjuntos de relaciones con restricciones de participación

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

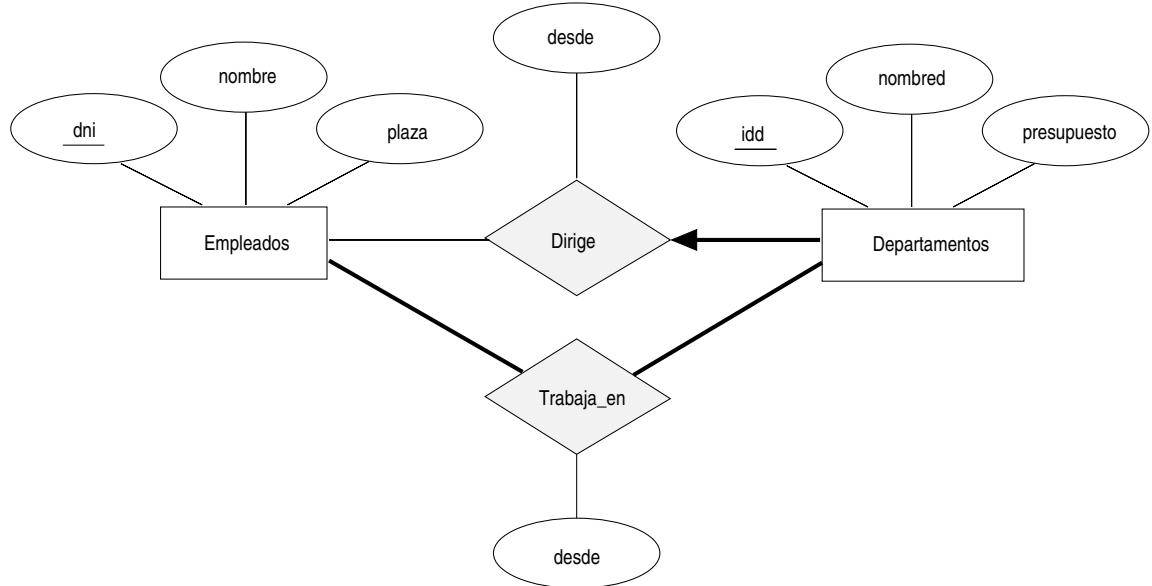


Figura 3.13 Dirige y Trabaja_en

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

```

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

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

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

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

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

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

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

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

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

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

3.5.5 Traducción de los conjuntos de entidades débiles

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

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

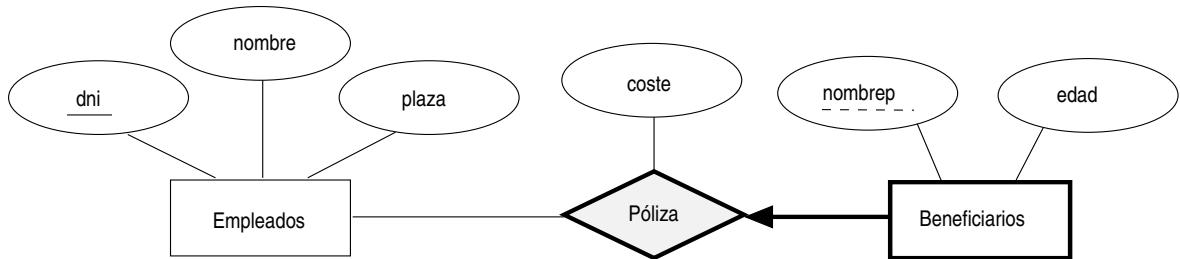


Figura 3.14 El conjunto de entidades débiles Beneficiarios

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

```

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

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

3.5.6 Traducción de las jerarquías de clase

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

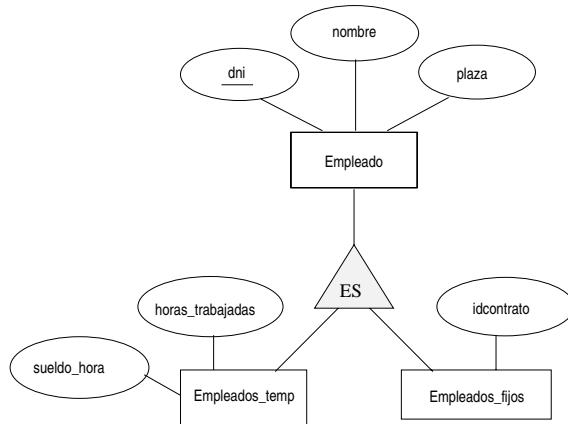


Figura 3.15 Jerarquía de clases

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

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

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

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

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

3.5.7 Traducción de los diagramas ER con agregación

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

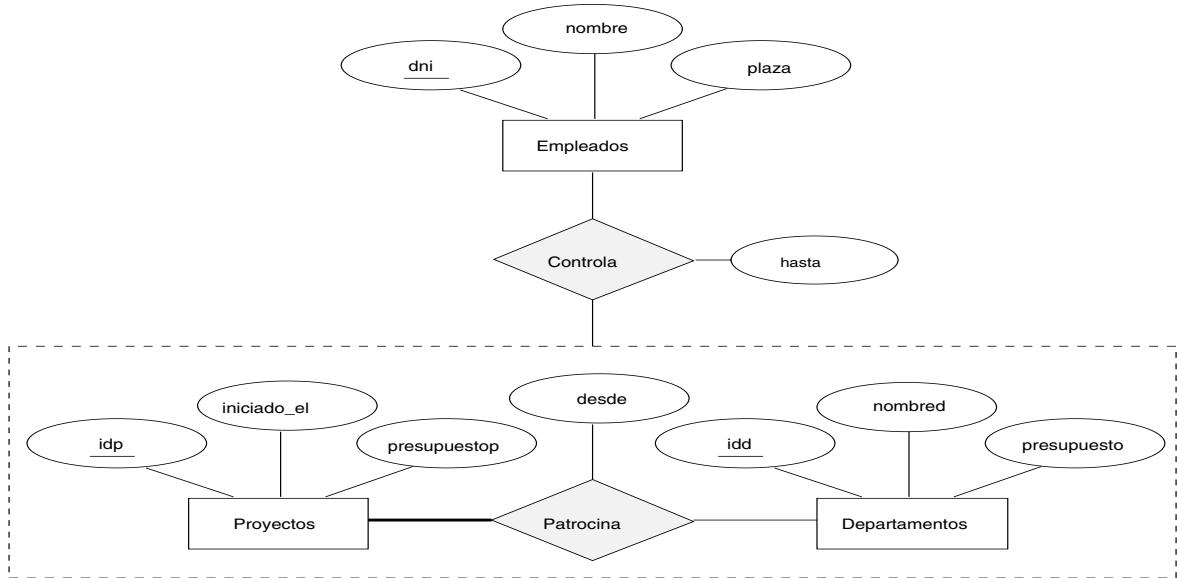


Figura 3.16 Agregación

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

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

1. Hay que registrar los atributos descriptivos (en este ejemplo, *desde*) de la relación Patrocina.
2. No todos los patrocinios tienen un controlador y, por tanto, puede que algunos pares $\langle idp, idd \rangle$ de la relación Patrocina no aparezcan en la relación Controla.

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

3.5.8 Del modelo ER al relacional: más ejemplos

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

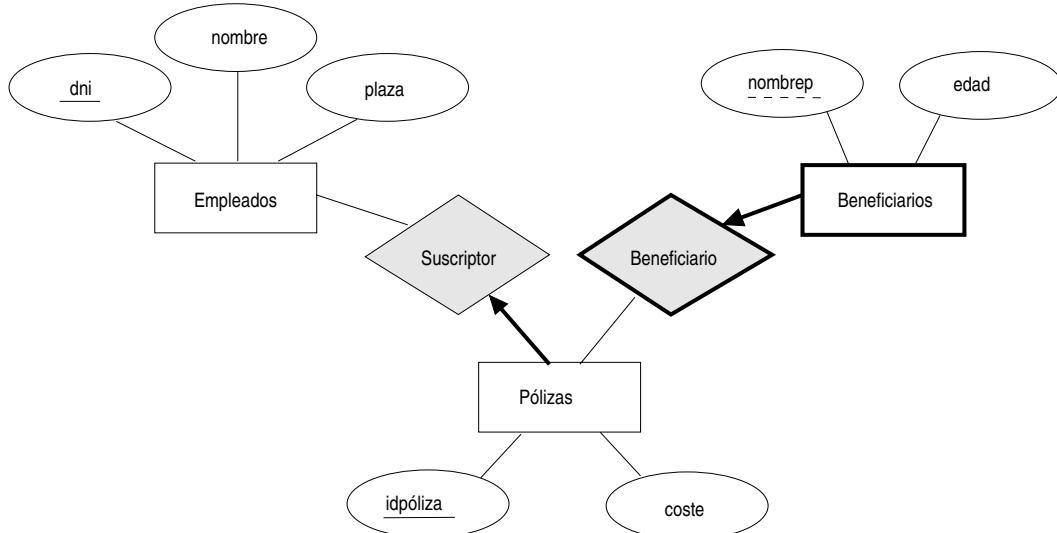


Figura 3.17 Las pólizas de nuevo

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

```

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

```

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

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

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

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

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

3.6 INTRODUCCIÓN A LAS VISTAS

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

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

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

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

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

<i>nombre</i>	<i>ide</i>	<i>asignatura</i>
Jiménez	53666	Historia105
García	53832	Reggae203

Figura 3.18 Un ejemplar de la vista Alumnos-Notable

3.6.1 Vistas, independencia de datos y seguridad

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

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

3.6.2 Actualizaciones de las vistas

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

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

utilizar operaciones de agregación⁴. Estas vistas se denominan **vistas actualizables**. Esta definición está simplificada en exceso, pero captura el espíritu de las restricciones. Siempre se puede implementar una actualización de estas vistas restringidas mediante la actualización de la tabla base subyacente de una manera no ambigua. Considérese la vista siguiente:

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

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

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

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

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

Necesidad de restringir las actualizaciones de las vistas

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

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

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

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

<i>nombrec</i>	<i>añoi</i>	<i>nombres</i>
Vela	1996	Díaz
Senderismo	1997	Sánchez
Remo	1998	Sánchez

Figura 3.19 El ejemplar *C* de Clubs

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

Figura 3.20 El ejemplar *A3* de Alumnos

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

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

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

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

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

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

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

Figura 3.21 Ejemplar de AlumnosActivos

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

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

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

3.7 ELIMINACIÓN Y MODIFICACIÓN DE TABLAS Y VISTAS

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

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

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

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

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

3.8 ESTUDIO DE UN CASO: LA TIENDA EN INTERNET

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

```

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

```

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

3.9 PREGUNTAS DE REPASO

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

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

92 Sistemas de gestión de bases de datos

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

EJERCICIOS

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Ejercicio 3.20 Considérese el esquema siguiente:

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

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

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



EJERCICIOS BASADOS EN UN PROYECTO

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

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

Ejercicio 3.23 ¿Qué restricciones de integridad hace cumplir Minibase?

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

NOTAS BIBLIOGRÁFICAS

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

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

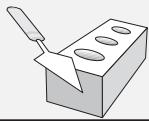
Una rica teoría sostiene el campo de las bases de datos relacionales. Entre los textos dedicados a los aspectos teóricos están los de Atzeni y DeAntonellis [29]; Maier [309]; y Abiteboul, Hull, y Vianu [1]. [265] es un excelente artículo recopilatorio.

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

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

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

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



4

ÁLGEBRA Y CÁLCULO RELACIONALES

- ☞ ¿Cuál es el fundamento de los lenguajes de consulta relacionales como SQL? ¿Cuál es la diferencia entre los lenguajes procedimentales y los declarativos?
- ☞ ¿Qué es el álgebra relacional y por qué es importante?
- ☞ ¿Cuáles son los operadores algebraicos básicos y cómo se combinan con las consultas complejas?
- ☞ ¿Qué es el cálculo relacional y por qué es importante?
- ☞ ¿Qué subconjunto de la lógica matemática se emplea en el cálculo relacional y cómo se utiliza para escribir consultas?
- ➡ **Conceptos fundamentales:** álgebra relacional, selección, proyección, unión, intersección, producto cartesiano, reunión, división; cálculo relacional de tuplas, cálculo relacional de dominios, fórmulas, cuantificadores universales y existenciales, variables ligadas y libres.

Mantente firme en tu negativa a permanecer consciente durante las clases de álgebra. En la vida real, te lo aseguro, no tropezarás con ella.

— Fran Lebowitz, *Estudios sociales*

En este capítulo se presentan dos lenguajes formales de consulta asociados con el modelo relacional. Los **lenguajes de consulta** son lenguajes especializados para formular preguntas, o **consultas**, que implican a los datos de una base de datos dada. Tras tratar algunos preliminares en el Apartado 4.1, se tratará el *álgebra relacional* en el Apartado 4.2. Las consultas se componen en el álgebra relacional mediante un conjunto de operadores, y cada consulta describe un procedimiento paso a paso para calcular la respuesta deseada; es decir, las consultas se especifican de manera *operativa*. En el Apartado 4.3 se trata el *cálculo relacional*, en el que cada consulta describe la respuesta deseada sin especificar la manera en que se debe calcular

la respuesta; este estilo de consulta no procedimental se denomina *declarativo*. Se suele hacer referencia al álgebra y al cálculo relacionales como álgebra y cálculo, respectivamente. Se comparará la potencia expresiva del álgebra y del cálculo en el Apartado 4.4. Estos lenguajes formales de consulta han influido enormemente en lenguajes de consulta comerciales como SQL, que se trata en capítulos posteriores.

4.1 PRELIMINARES

Se comenzará por aclarar algunos aspectos importantes de las consultas relacionales. Tanto las entradas como los resultados de las consultas son relaciones. Las consultas se evalúan a partir de *ejemplares* de cada relación de entrada y genera un ejemplar de la relación de salida. En el Apartado 3.4 se emplearon nombres de campos para hacer referencia a los campos porque esa notación hace más legibles las consultas. Una alternativa es relacionar siempre los campos de una relación dada en el mismo orden y hacer referencia a ellos por su posición en vez de por su nombre.

Al definir el álgebra y el cálculo relacionales, la alternativa de hacer referencia a los campos por su posición resulta más conveniente que hacerlo por su nombre: las consultas suelen suponer el cálculo de resultados intermedios, que son en sí mismos ejemplares de relaciones; y, si se empleara el nombre para hacer referencia a los campos, la definición de las estructuras del lenguaje de consulta debería especificar el nombre de los campos para todos los ejemplares de las relaciones intermedias. Esto puede resultar tedioso y realmente es un aspecto secundario, ya que de todas maneras se puede hacer referencia a los campos por su posición. Por otro lado, el nombre de los campos hace las consultas más legibles.

Debido a estas consideraciones se emplea la notación posicional para definir formalmente el álgebra y el cálculo relacionales. Por comodidad, también se introducirán convenios sencillos que permitan que las relaciones intermedias “hereden” los nombres de los campos.

Se presentarán varias consultas de ejemplo mediante el esquema siguiente:

Marineros(idm: integer, nombrem: string, categoría: integer, edad: real)

Barcos(idb: integer, nombreb: string, color: string)

Reservas(idm: integer, idb: integer, fecha: date)

Los campos de la clave están subrayados, y el dominio de cada campo se indica después de su nombre. Por tanto, *idm* es la clave de Marineros, *idb* es la de Barcos y los tres campos de Reservas forman conjuntamente su clave. Se hará referencia a los campos de cada ejemplar de cualquiera de estas relaciones por el nombre, o posicionalmente, empleando el nombre en el que se acaban de mencionar.

En varios ejemplos que ilustran los operadores del álgebra relacional, se emplean los ejemplares *M1* y *M2* (de Marineros) y *R1* (de Reservas), que pueden verse en las Figuras 4.1, 4.2 y 4.3, respectivamente.

4.2 ÁLGEBRA RELACIONAL

El álgebra relacional es uno de los dos lenguajes formales de consulta asociados con el modelo relacional. Las consultas en el álgebra se componen mediante un conjunto de operadores. Una propiedad fundamental es que todos los operadores del álgebra aceptan (uno o dos) ejem-

<i>idm</i>	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>
22	Domínguez	7	45,0
31	López	8	55,5
58	Rubio	10	35,0

Figura 4.1 Ejemplar *M1* de Marineros

<i>idm</i>	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>
28	Yiyo	9	35,0
31	Lorca	8	55,5
44	Gallardo	5	35,0
58	Rubio	10	35,0

Figura 4.2 Ejemplar *M2* de Marineros

<i>idm</i>	<i>idb</i>	<i>fecha</i>
22	101	10/10/96
58	103	11/12/96

Figura 4.3 Ejemplar *R1* de Reservas

plares de relaciones como argumentos y devuelven un ejemplar de relación como resultado. Esta propiedad facilita la *composición* de operadores para formar consultas complejas —las **expresiones del álgebra relacional** se definen recursivamente como relaciones, como operadores unarios del álgebra aplicados a una sola expresión o como operadores binarios del álgebra aplicados a dos expresiones—. En los apartados siguientes se describirán los operadores básicos del álgebra (selección, proyección, unión, producto cartesiano y diferencia), así como algunos operadores adicionales que se pueden definir en términos de los operadores básicos pero aparecen con la suficiente frecuencia como para merecer atención especial.

Cada consulta relacional describe un procedimiento detallado para el cálculo de la respuesta deseada, siguiendo el orden en el que se aplican los operadores en la consulta. La naturaleza procedimental del álgebra permite considerar cada expresión algebraica como una receta, o plan, para la evaluación de una consulta, y los sistemas relacionales, de hecho, emplean expresiones algebraicas para representar los planes de evaluación de las consultas.

4.2.1 Selección y proyección

El álgebra relacional incluye operadores para *seleccionar* filas de las relaciones (σ) y para *proyectar* columnas (π). Estas operaciones permiten manipular los datos de una sola relación. Considérese el ejemplar de la relación Marineros de la Figura 4.2, denotada como *M2*. Se pueden recuperar las filas correspondientes a marineros expertos mediante el operador σ . La expresión

$$\sigma_{categoría>8}(M2)$$

da como resultado la relación de la Figura 4.4. El subíndice *categoría>8* especifica el criterio de selección que hay que aplicar al recuperar tuplas.

El operador selección σ especifica las tuplas que hay que conservar mediante una *condición de selección*. En general, la condición de selección es una combinación booleana (es decir, una expresión que emplea los conectores lógicos \wedge y \vee) de *términos* que tienen la forma *atributo op constante* o *atributo1 op atributo2*, donde *op* es uno de los operadores de comparación

<i>idm</i>	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>
28	Yiyo	9	35,0
58	Rubio	10	35,0

Figura 4.4 $\sigma_{categoría > 8}(M2)$

<i>nombrem</i>	<i>categoría</i>
Yiyo	9
Lorca	8
Gallardo	5
Rubio	10

Figura 4.5 $\pi_{nombrem, categoría}(M2)$

$<, \leq, =, \neq, \geq >$ o $>$. La referencia a los atributos puede hacerse por su posición (de la forma *.i* o *i*) o por su nombre (de la forma *.nombre* o *nombre*). El esquema del resultado de una selección es el esquema del ejemplar de la relación de entrada.

El operador proyección π permite extraer columnas de las relaciones; por ejemplo, se puede averiguar el nombre y la categoría de todos los marineros mediante π . La expresión

$$\pi_{nombrem, categoría}(M2)$$

de como resultado la relación de la Figura 4.5. El subíndice *nombrem, categoría* especifica los campos que hay que conservar; los otros campos se “descartan”. El esquema del resultado de cada proyección está formado por los campos proyectados.

Supóngase que únicamente se desea averiguar la edad de los marineros. La expresión

$$\pi_{edad}(M2)$$

de como resultado la relación de la Figura 4.6. Lo importante es que, aun permite tres marineros tienen treinta y cinco años de edad, sólo aparece una tupla con *edad=35,0* en el resultado de la proyección. Esto se debe a la definición de las relaciones como *conjuntos* de tuplas. En la práctica, los sistemas reales suelen omitir el costoso paso de eliminar las *tuplas duplicadas*, lo que lleva a relaciones que son multiconjuntos. Sin embargo, el estudio del álgebra y del cálculo relacionales da por supuesto que siempre se lleva a cabo la eliminación de duplicados, de modo que las relaciones siempre son conjuntos de tuplas.

Dado que el resultado de las expresiones del álgebra relacional siempre es una relación, se puede colocar una expresión en cualquier lugar en que se espere una relación. Por ejemplo, se puede calcular el nombre y la categoría de los marineros de mayor categoría combinando dos de las consultas anteriores. La expresión

$$\pi_{nombrem, categoría}(\sigma_{categoría > 8}(M2))$$

produce el resultado que puede verse en la Figura 4.7. Se obtiene aplicando la selección a *M2* (para conseguir la relación mostrada en la Figura 4.4) y aplicando luego la proyección.

<i>edad</i>
35,0
55,5

Figura 4.6 $\pi_{edad}(M2)$

<i>nombrem</i>	<i>categoría</i>
Yiyo	9
Rubio	10

Figura 4.7 $\pi_{nombrem, categoría}(\sigma_{categoría > 8}(M2))$

4.2.2 Operaciones con conjuntos

Las siguientes operaciones estándar con conjuntos también están disponibles en el álgebra relacional: *unión* (\cup), *intersección* (\cap), *diferencia de conjuntos* ($-$) y *producto cartesiano* (\times).

- **Unión.** $R \cup S$ devuelve el ejemplar de una relación que contiene todas las tuplas que aparecen *bien* en algún ejemplar de la relación R , en algún ejemplar de la relación S (o en algún ejemplar de ambas). R y S deben ser *compatibles para la unión*, y se define que el esquema del resultado es idéntico al de R (o S).

Se dice que dos ejemplares de relaciones son **compatibles para la unión** si se cumplen las condiciones siguientes:

- tienen el mismo número de campos y
- los campos correspondientes, tomados en orden de izquierda a derecha, tienen los mismos *dominios*.

Obsérvese que el nombre de los campos no se utiliza en la definición de compatibilidad para la unión. Por comodidad se supondrá que los campos de $R \cup S$ heredan los nombres de R , si es que los campos de R tienen nombre. (Esta suposición queda implícita en que el esquema de $R \cup S$ será idéntico al de R , tal y como se ha afirmado.)

- **Intersección.** $R \cap S$ devuelve el ejemplar de una relación que contiene todas las tuplas que aparecen *tanto* en R como en S . Las relaciones R y S deben ser compatibles para la unión, y se define que el esquema del resultado es idéntico al de R .
- **Diferencia de conjuntos.** $R - S$ devuelve el ejemplar de una relación que contiene todas las tuplas que aparecen en R pero no en S . Las relaciones R y S deben ser compatibles para la unión y se define que el esquema del resultado es idéntico al de R .
- **Producto cartesiano.** $R \times S$ devuelve el ejemplar de una relación cuyo esquema contiene todos los campos de R (en el mismo orden en el que aparecen en R) seguidos por todos los campos de S (en el mismo orden en que aparecen en S). El resultado de $R \times S$ contiene la tupla $\langle r, s \rangle$ (la concatenación de las tuplas r y s) por cada par de tuplas $r \in R$, $s \in S$. La operación producto cartesiano se denomina a veces **producto aspa**.

En este libro se aplica el convenio de que los campos de $R \times S$ heredan el nombre de los campos correspondientes de R y de S . Es posible que tanto R como S contengan campos que tengan el mismo nombre; esta situación crea un *conflicto de denominaciones*. Los campos correspondientes de $R \times S$ quedan sin nombre y se hace referencia a ellos solamente por su posición.

Obsérvese en las definiciones anteriores que cada operador puede aplicarse a ejemplares de relaciones que se calculan mediante (sub)expresiones del álgebra relacional.

Ahora se ilustrarán estas definiciones mediante varios ejemplos. La unión de $M1$ y $M2$ puede verse en la Figura 4.8. Los campos se relacionan en orden; el nombre de los campos también se hereda de $M1$. $M2$ tiene los mismos nombres de campos, por supuesto, ya que

también es un ejemplar de Marineros. En general, los campos de $M2$ pueden tener nombres diferentes; recuérdese que sólo se exige que los dominios coincidan. Obsérvese que el resultado es un *conjunto* de tuplas. Las tuplas que aparecen tanto en $M1$ como en $M2$ sólo aparecen una vez en $M1 \cup M2$. Además, $M1 \cup R1$ no es una operación válida, ya que las dos relaciones no son compatibles para la unión. La intersección de $M1$ y de $M2$ puede verse en la Figura 4.9, y la diferencia de conjuntos $M1 - M2$ en la Figura 4.10.

<i>idm</i>	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>
22	Domínguez	7	45,0
31	López	8	55,5
58	Rubio	10	35,0
28	Yiyo	9	35,0
44	Gallardo	5	35,0

Figura 4.8 $M1 \cup M2$

<i>idm</i>	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>
31	López	8	55,5
58	Rubio	10	35,0

Figura 4.9 $M1 \cap M2$

<i>idm</i>	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>
22	Domínguez	7	45,0

Figura 4.10 $M1 - M2$

El resultado del producto cartesiano $M1 \times R1$ puede verse en la Figura 4.11. Como tanto $R1$ como $M1$ tienen un campo denominado *idm*, por el convenio establecido para los nombres de los campos, los dos campos correspondientes de $M1 \times R1$ permanecen sin nombre, y se hace referencia a ellos únicamente por la posición en la que aparecen en la Figura 4.11. Los campos de $M1 \times R1$ tienen los mismo dominios que los correspondientes de $R1$ y de $M1$. En la Figura 4.11 *idm* aparece entre paréntesis para enfatizar que no es un nombre de campo heredado; sólo se hereda el dominio correspondiente.

(<i>idm</i>)	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>	(<i>idm</i>)	<i>ldb</i>	<i>fecha</i>
22	Domínguez	7	45,0	22	101	10/10/96
22	Domínguez	7	45,0	58	103	11/12/96
31	López	8	55,5	22	101	10/10/96
31	López	8	55,5	58	103	11/12/96
58	Rubio	10	35,0	22	101	10/10/96
58	Rubio	10	35,0	58	103	11/12/96

Figura 4.11 $M1 \times R1$

4.2.3 Renombramiento

Se ha tenido cuidado en adoptar convenios de denominación para los campos de forma que garanticen que el resultado de las expresiones del álgebra relacional herede de manera natural el nombre de los campos de su relación de entrada siempre que sea posible. Sin embargo, pueden surgir conflictos de denominación en algunos casos; por ejemplo, en $M1 \times R1$. Por tanto, resulta conveniente poder dar nombre de manera explícita a los campos de cada ejemplar de las relaciones definidas por expresiones del álgebra relacional. De hecho, suele resultar conveniente ponerle nombre al propio ejemplar, de modo que se puedan descomponer expresiones algebraicas largas en fragmentos más pequeños dando nombre a los resultados de las subexpresiones.

Con este objetivo se introduce el operador **renombramiento** ρ . La expresión $\rho(R(\bar{F}), E)$ toma una expresión arbitraria del álgebra relacional E y devuelve un ejemplar de una relación (nueva) denominada R . R contiene las mismas tuplas que el resultado de E , y tiene el mismo esquema que E , pero se cambia el nombre de algunos campos. El nombre de los campos de la relación R es el mismo que en E , salvo para los campos que se han cambiado de nombre en la lista de renombramientos \bar{F} , que es una lista de términos que tiene la forma *nombreatiguo* \rightarrow *nombrenuevo*, o bien *posición* \rightarrow *nombrenuevo*. Para que ρ esté bien definida, las referencias a los campos (de la forma *nombresantiguos* o *posiciones* en la lista de renombramientos) no pueden ser ambiguas, y no puede haber dos campos del resultado que tengan el mismo nombre. A veces sólo se desea cambiar el nombre de campos o (re)nombrar la relación; por tanto, se considera que tanto R como \bar{F} son opcionales en el empleo de ρ . (Por supuesto, no tiene sentido omitir las dos.)

Por ejemplo, la expresión $\rho(C(1 \rightarrow idm1, 5 \rightarrow idm2), M1 \times R1)$ devuelve una relación que contiene las tuplas de la Figura 4.11 y tiene el esquema siguiente: $C(idm1: integer, nombrem: string, categoría: integer, edad: real, idm2: integer, idb: integer, fecha: dates)$.

Es habitual incluir algunos operadores adicionales en el álgebra, pero todos ellos se pueden definir en términos de los operadores ya definidos. (De hecho, el operador renombramiento sólo se necesita por comodidad sintáctica, e incluso el operador \cap es redundante; $R \cap M$ se puede definir como $R - (R - M)$.) Estos operadores adicionales y su definición en términos de los operadores básicos se toman en consideración en los dos subapartados siguientes.

4.2.4 Reuniones (*join*)

La operación *reunión* es una de las más útiles del álgebra relacional y la empleada más frecuentemente para combinar información de dos o más relaciones. Aunque las reuniones se pueden definir como productos cartesianos seguidos de selecciones y proyecciones, aparecen en la práctica mucho más frecuentemente que los meros productos cartesianos. Además, el resultado de un producto cartesiano suele ser mucho más grande que el de una reunión, y resulta muy importante reconocer las reuniones e implementarlas sin materializar el producto cartesiano subyacente (aplicando las selecciones y proyecciones “sobre la marcha”). Por estas

razones, las reuniones han recibido mucha atención, y hay diversas variantes de la operación reunión¹.

Reuniones condicionales

La versión más general de la operación reunión acepta una *condición de reunión* c y un par de ejemplares de relaciones como argumentos y devuelve el ejemplar de una relación. La *condición de reunión* es idéntica en su forma a las *condiciones de selección*. La operación se define de la manera siguiente:

$$R \bowtie_c S = \sigma_c(R \times S)$$

Así, se define que \bowtie es un producto cartesiano seguido de una selección. Obsérvese que la condición c puede hacer referencia a atributos tanto de R como de S (y suele *hacerlo*). La referencia a los atributos de las relaciones, por ejemplo R , puede hacerse por su posición (de la forma $R.i$) o por su nombre (de la forma $R.nombre$).

A modo de ejemplo, el resultado de $M1 \bowtie_{M1.idm < R1.idm} R1$ puede verse en la Figura 4.12. Como idm aparece tanto en $M1$ como en $R1$, los campos correspondientes del resultado del producto cartesiano $M1 \times R1$ (y, por tanto, del resultado de $M1 \bowtie_{M1.idm < R1.idm} R1$) se quedan sin nombre. Los dominios se heredan de los campos correspondientes de $M1$ y de $R1$.

(idm)	$nombrerem$	$categoria$	$edad$	(idm)	idb	$fecha$
22	Domínguez	7	45,0	58	103	11/12/96
31	López	8	55,5	58	103	11/12/96

Figura 4.12 $M1 \bowtie_{M1.idm < R1.idm} R1$

Equirreuniones

Un caso especial frecuente de la operación reunión $R \bowtie S$ es que la *condición de reunión* sólo consista en igualdades (conectadas por \wedge) de la forma $R.nombre1 = S.nombre2$, es decir, igualdades entre dos campos de R y de S . En este caso, se produce evidentemente cierta redundancia al conservar en el resultado los dos atributos. Para condiciones de reunión que sólo contienen igualdades de este tipo, se puede refinar la operación de reunión realizando una proyección adicional en la que se descarte $S.nombre2$. La operación reunión con este refinamiento se denomina **equirreunión**.

El esquema del resultado de las equirreuniones contiene los campos de R (con los mismos nombres y dominios que en R) seguidos de los campos de S que no aparecen en las condiciones de reunión. Si este conjunto de campos de la relación resultante incluye dos campos que hereden el mismo nombre de R y de S , quedarán sin nombre en la relación resultante.

En la Figura 4.13 se ilustra $M1 \bowtie_{M1.idm = R1.idm} R1$. Obsérvese que en el resultado sólo aparece un campo denominado idm .

¹En este capítulo no se estudian las diversas variantes de las reuniones. En el Capítulo 5 se trata una clase importante de reuniones, denominadas *reuniones externas*.

<i>idm</i>	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>	<i>idb</i>	<i>fecha</i>
22	Domínguez	7	45,0	101	10/10/96
58	Rubio	10	35,0	103	11/12/96

Figura 4.13 $M1 \bowtie_{M1.idm=R1.idm} R1$

Reunión natural

Otro caso especial de la operación reunión $R \bowtie S$ es la equirreunión en la que las igualdades se especifican para *todos* los campos que tienen el mismo nombre en R y en S . En este caso, se puede omitir simplemente la condición de reunión; el valor predeterminado es que la condición de reunión sea un conjunto de igualdades para todos los campos comunes. Este caso especial se denomina *reunión natural*, y tiene la propiedad de garantizar que el resultado no tenga dos campos con el mismo nombre.

La expresión de la equirreunión $M1 \bowtie_{R.idm=M.idm} R1$ es realmente una operación natural, y se puede denotar simplemente como $M1 \bowtie R1$, ya que el único campo común es *idm*. Si las dos relaciones no tienen atributos en común, $M1 \bowtie R1$ no es más que el producto cartesiano.

4.2.5 División

El operador división resulta útil para expresar determinadas clases de consultas como, por ejemplo, “Averiguar el nombre de los marineros que tienen reservados todos los barcos”. Comprender la manera de emplear los operadores básicos del álgebra para definir la división es un ejercicio útil. Sin embargo, el operador división no tiene la misma importancia que los demás operadores —no hace falta tan a menudo, y los sistemas de bases de datos no intentan aprovechar la semántica de la división implementándola como un operador diferente (como, por ejemplo, se hace con el operador reunión)—.

Se tratará la división mediante un ejemplo. Considérense dos ejemplares de relación, A y B , de las que A tiene (exactamente) dos campos, x e y , y B tiene sólo un campo, y , con el mismo dominio que en A . Se define la operación *división* A/B como el conjunto de todos los valores de x (en forma de tuplas unarias) tales que, para *todos* los valores de y de (cualquier tupla de) B , hay una tupla $\langle x,y \rangle$ de A .

Otra manera de comprender la división es la siguiente. Para cada valor de x de (la primera columna de) A , considérese el conjunto de valores de y que aparecen en (el segundo campo de) las tuplas de A con ese valor de x . Si ese conjunto contiene (todos los valores de y de) B , el valor de x se halla en el resultado de A/B .

Una analogía con la división de enteros puede resultar también de ayuda para comprender la división. Para los enteros A y B , A/B es el mayor entero Q tal que $Q * B \leq A$. Para los ejemplares de relaciones A y B , A/B es el mayor ejemplar de relación Q tal que $Q \times B \subseteq A$.

La división se ilustra en la Figura 4.14. Resulta de ayuda pensar en A como en una relación que enumera los repuestos facilitados por diferentes los proveedores y en las relaciones B como en listados de repuestos. A/B_i calcula los proveedores que facilitan *todos* los repuestos que aparecen en el ejemplar de relación B_i .

A	<i>idp</i>	<i>idr</i>	B1	<i>idr</i>	r2	A/B1	<i>idp</i>	p1	p2	p3	p4
p1	r1										
p1	r2		B2	<i>idr</i>	r2						
p1	r3				r4						
p1	r4										
p2	r1		B3	<i>idr</i>							
p2	r2				r1						
p3	r2				r2						
p4	r2				r4						
p4	r4										

A/B2	<i>idp</i>	p1	p4

A/B3	<i>idp</i>	p1

Figura 4.14 Ejemplos que ilustran la división

Expresar A/B en términos de los operadores algebraicos básicos es un ejercicio interesante, y el lector debería intentar hacerlo antes de seguir leyendo. La idea básica es calcular todos los valores de x de A que no estén *descalificados*. Un valor de x queda *descalificado* si, al adjuntarle un valor de y de B , se obtiene una tupla $\langle x,y \rangle$ que no está en A . Se pueden calcular las tuplas descalificadas mediante la expresión algebraica

$$\pi_x((\pi_x(A) \times B) - A)$$

Por tanto, se puede definir A/B como

$$\pi_x(A) - \pi_x((\pi_x(A) \times B) - A)$$

Para comprender la operación división en toda su generalidad hay que considerar el caso en que tanto x como y se sustituyen por un conjunto de atributos. La generalización es directa y se deja como ejercicio para el lector. Más adelante en este apartado se discutirán dos ejemplos más que ilustran la división (Consultas C9 y C10).

4.2.6 Más ejemplos de consultas algebraicas

Ahora se presentarán varios ejemplos para ilustrar la manera de escribir consultas en el álgebra relacional. Se empleará el esquema Marineros, Reservas y Barcos para todos los ejemplos de este apartado. Se emplearán los paréntesis como sea necesario para lograr que las expresiones algebraicas no resulten ambiguas. Obsérvese que todas las consultas de ejemplo de este capítulo reciben un único número de consulta. Estos números son únicos en este capítulo y en el capítulo sobre consultas de SQL (Capítulo 5). Esta numeración facilita la identificación de las consultas cuando se vuelven a examinar en el contexto del cálculo relacional y de SQL, y para comparar las diferentes maneras de escribir la misma consulta. (Todas las referencias a cada consulta se pueden hallar en el índice.)

En el resto de este capítulo (y en el Capítulo 5) se ilustran las consultas empleando los ejemplares *M3* de Marineros, *R2* de Reservas y *B1* de Barcos, que pueden verse en las Figuras 4.15, 4.16 y 4.17, respectivamente.

<i>idm</i>	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>
22	Domínguez	7	45,0
29	Bravo	1	33,0
31	López	8	55,5
32	Alández	8	25,5
58	Rubio	10	35,0
64	Horacio	7	35,0
71	Zuazo	10	16,0
74	Horacio	9	35,0
85	Arturo	3	25,5
95	Benito	3	63,5

Figura 4.15 El ejemplar *M3* de Marineros

<i>idm</i>	<i>idb</i>	<i>fecha</i>
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figura 4.16 El ejemplar *R2* de Reservas

(C1) Averiguar el nombre de los marineros que han reservado el barco 103.

Esta consulta se puede escribir de la manera siguiente:

$$\pi_{nombrem}(\sigma_{idb=103}(Reservas) \bowtie Marineros)$$

En primer lugar se calcula un conjunto de tuplas de Reservas con *idb* = 103 y luego se toma la reunión natural de este conjunto con Marineros. Esta expresión se puede evaluar con ejemplares de Reservas y de Marineros. Evaluada con los ejemplares *R2* y *M3*, da una relación que contiene un solo campo, denominado *nombrem*, y tres tuplas $\langle\text{Domínguez}\rangle$, $\langle\text{Horacio}\rangle$ y $\langle\text{López}\rangle$. (Obsérvese que hay dos marineros llamados Horacio y sólo uno de ellos ha reservado un barco rojo.)

<i>idb</i>	<i>nombreb</i>	<i>color</i>
101	Intrépido	azul
102	Intrépido	rojo
103	Campeón	verde
104	Místico	rojo

Figura 4.17 El ejemplar *B1* de Barcos

Esta consulta se puede descomponer en fragmentos más pequeños mediante el operador renombramiento ρ :

$$\begin{aligned} & \rho(Temp1, \sigma_{idb=103}(Reservas)) \\ & \rho(Temp2, Temp1 \bowtie Marineros) \\ & \pi_{nombreb}(Temp2) \end{aligned}$$

Obsérvese que, como sólo se está empleando ρ para dar nombre a las relaciones intermedias, la lista de renombramientos es opcional y se omite. *Temp1* denota una relación intermedia que identifica las reservas del barco 103. *Temp2* es otra relación intermedia, y denota los marineros

que han hecho alguna reserva del conjunto $Temp1$. Los ejemplares de estas relaciones, al evaluar esta consulta con los ejemplares $R2$ y $M3$, se ilustran en las Figuras 4.18 y 4.19. Finalmente se extrae la columna $nombrem$ de $Temp2$.

<i>idm</i>	<i>idb</i>	<i>fecha</i>
22	103	10/8/98
31	103	11/6/98
74	103	9/8/98

Figura 4.18 Ejemplar de $Temp1$

<i>idm</i>	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>	<i>idb</i>	<i>fecha</i>
22	Domínguez	7	45,0	103	10/8/98
31	López	8	55,5	103	11/6/98
74	Horacio	9	35,0	103	9/8/98

Figura 4.19 Ejemplar de $Temp2$

La versión de la consulta que emplea ρ es básicamente igual que la consulta original; el empleo de ρ no es más que una simplificación sintáctica. Sin embargo, realmente hay diferentes maneras de escribir las consultas en el álgebra relacional. He aquí otro modo de escribir esta consulta:

$$\pi_{nombrem}(\sigma_{idb=103}(Reservas \bowtie Marineros))$$

En esta versión se calcula en primer lugar la reunión natural de Reservas y Marineros y luego se aplican la selección y la proyección.

Este ejemplo permite vislumbrar el papel desempeñado por el álgebra en los SGBD relationales. Los usuarios expresan las consultas en un lenguaje como SQL. El SGBD traduce las consultas de SQL a (una versión ampliada del) álgebra relacional y luego busca otras expresiones algebraicas que produzcan las mismas respuestas pero que sean más económicas de evaluar. Si la consulta del usuario se traduce primero en la expresión

$$\pi_{nombrem}(\sigma_{idb=103}(Reservas \bowtie Marineros))$$

un buen optimizador de consultas hallará la expresión equivalente

$$\pi_{nombrem}(\sigma_{idb=103}(Reservas) \bowtie Marineros)$$

Además, el optimizador reconocerá que es probable que la segunda expresión sea menos costosa de calcular, ya que el tamaño de las relaciones intermedias es menor, gracias al empleo temprano de la selección.

(C2) Averiguar el nombre de los marineros que han reservado barcos rojos.

$$\pi_{nombrem}(\sigma_{color='rojo'}(Barcos) \bowtie Reservas \bowtie Marineros)$$

Esta consulta implica una serie de dos reuniones. En primer lugar, se escogen (tuplas que describan) barcos rojos. Luego se reúne este conjunto con Reservas (reunión natural, con la igualdad especificada para la columna idb) para identificar las reservas de barcos rojos. Luego se reúne la relación intermedia resultante con Marineros (reunión natural, con la igualdad especificada para la columna idm) para recuperar el nombre de los marineros que han hecho reservas de barcos rojos. Finalmente se proyectan los nombres de los marineros. La respuesta, cuando se evalúa con los ejemplares $B1$, $R2$ y $M3$, contiene los nombres Domínguez, Horacio y López. (Obsérvese que hay dos marineros llamados Horacio y sólo uno de ellos ha reservado un barco rojo.)

Una expresión equivalente es:

$$\pi_{nombrem}(\pi_{idm}(\pi_{idb}(\sigma_{color='rojo'}(Barcos)) \bowtie Reservas) \bowtie Marineros)$$

Se invita al lector a reescribir estas dos consultas empleando ρ para hacer explícitas las relaciones intermedias y a comparar los esquemas de las relaciones intermedias. La segunda expresión genera relaciones intermedias con menos campos (y, por tanto, también es probable que dé lugar a ejemplares de las relaciones intermedias con menos tuplas). Un optimizador de consultas relacionales intentaría llegar a la segunda expresión si se le diera la primera.

(C3) Averiguar el color de los barcos reservados por López.

$$\pi_{color}(\sigma_{nombrem='López'}(Marineros) \bowtie Reservas \bowtie Barcos)$$

Esta consulta es muy parecida a la que se empleó para calcular los marineros que habían reservado barcos rojos. Para los ejemplares $B1$, $R2$ y $M3$ la consulta devuelve los colores verde y rojo.

(C4) Averiguar el nombre de los marineros que han reservado, como mínimo, un barco.

$$\pi_{nombrem}(Marineros \bowtie Reservas)$$

La reunión de Marineros y Reservas crea una relación intermedia en la que las tuplas consisten en una tupla de Marineros “adjunta a” otra de Reservas. Sólo aparece una tupla de Marineros en (alguna tupla de) esta relación intermedia si, como mínimo, una tupla de Reservas tiene el mismo valor de idm , es decir, si ese marinero ha hecho alguna reserva. La respuesta, cuando se evalúa para los ejemplares $B1$, $R2$ y $M3$, contiene las tres tuplas $\langle\text{Domínguez}\rangle$, $\langle\text{Horacio}\rangle$ y $\langle\text{López}\rangle$. Aunque dos marineros llamados Horacio hayan reservado un barco, la respuesta sólo contiene una copia de la tupla $\langle\text{Horacio}\rangle$, ya que es una *relación*, es decir, un *conjunto* de tuplas, sin duplicados.

En este momento merece la pena destacar la frecuencia con que la reunión natural se emplea en estos ejemplos. Esta frecuencia es más que una mera coincidencia basada en el conjunto de consultas que se ha decidido estudiar; la reunión natural es una operación muy natural, ampliamente utilizada. En especial, la reunión natural se emplea a menudo para reunir dos tablas según un campo que forme clave externa. En la Consulta C4, por ejemplo, la reunión iguala los campos idm de Marineros y Reservas, y el campo idm de Reservas es una clave externa que hace referencia al campo idm de Marineros.

(C5) Averiguar el nombre de los marineros que han reservado barcos rojos o verdes.

$$\begin{aligned} &\rho(Barcostemp, \sigma_{color='rojo'}(Barcos) \cup \sigma_{color='verde'}(Barcos)) \\ &\pi_{nombrem}(Barcostemp \bowtie Reservas \bowtie Marineros) \end{aligned}$$

Se identifica el conjunto de todos los barcos que son rojos o verdes (Barcostemp, que contiene los barcos con los $idbs$ 102, 103 y 104 de los ejemplares $B1$, $R2$ y $M3$). Luego se reúne con Reservas para identificar los $idms$ de los marineros que han reservado alguno de esos barcos; esto no da los $idms$ 22, 31, 64 y 74 para los ejemplares del ejemplo. Finalmente, se reúne (una relación intermedia que contiene este conjunto de $idms$) con Marineros para averiguar el nombre de los Marineros con esas $idms$. Esto resulta en los nombres Domínguez,

Horacio y López para los ejemplares *B1*, *R2* y *M3*. Otra definición equivalente es la siguiente:

$$\begin{aligned} & \rho(\text{Barcostemp}, \sigma_{\text{color}='rojo' \vee \text{color}='verde'}(\text{Barcos})) \\ & \pi_{\text{nombrem}}(\text{Barcostemp} \bowtie \text{Reservas} \bowtie \text{Marineros}) \end{aligned}$$

Consideremos ahora una consulta muy parecida.

(C6) *Averiguar el nombre de los marineros que han reservado barcos rojos y verdes.* Resulta tentador intentar hacer esto sustituyendo simplemente \cup por \cap en la definición de Barcostemp:

$$\begin{aligned} & \rho(\text{Barcostemp2}, \sigma_{\text{color}='rojo'}(\text{Barcos}) \cap (\sigma_{\text{color}='verde'}(\text{Barcos})) \\ & \pi_{\text{nombrem}}(\text{Barcostemp2} \bowtie \text{Reservas} \bowtie \text{Marineros}) \end{aligned}$$

Sin embargo, esta solución es incorrecta —en vez de lo que se pretende, intenta calcular los marineros que han reservado barcos que sean a la vez rojos y verdes—. (Dado que *idb* es una clave de Barcos, cada barco sólo puede tener un color; esta consulta devuelve siempre un conjunto de respuestas vacío.) El enfoque correcto es averiguar los marineros que han reservado barcos rojos, luego los que han reservado barcos verdes y, más tarde, tomar la intersección de esos dos conjuntos:

$$\begin{aligned} & \rho(\text{Rojotemp}, \pi_{\text{idm}}(\sigma_{\text{color}='rojo'}(\text{Barcos}) \bowtie \text{Reservas})) \\ & \rho(\text{Verdetemp}, \pi_{\text{idm}}(\sigma_{\text{color}='verde'}(\text{Barcos}) \bowtie \text{Reservas})) \\ & \pi_{\text{nombrem}}((\text{Rojotemp} \cap \text{Verdetemp}) \bowtie \text{Marineros}) \end{aligned}$$

Las dos relaciones temporales calculan los *idms* de los marineros, y su intersección identifica los marineros que han reservado tanto barcos rojos como barcos verdes. En los ejemplares *B1*, *R2* y *M3*, los *idms* de los marineros que han reservado barcos rojos son 22, 31 y 64. Los *idms* de los marineros que han reservado barcos verdes son 22, 31 y 74. Por tanto, los marineros 22 y 31 han reservado tanto barcos rojos como barcos verdes; sus nombres son Domínguez y López.

Esta formulación de la Consulta C6 se puede adaptar fácilmente para averiguar los marineros que han reservado barcos rojos *o* verdes (Consulta C5); basta con sustituir \cap por \cup :

$$\begin{aligned} & \rho(\text{Rojotemp}, \pi_{\text{idm}}(\sigma_{\text{color}='rojo'}(\text{Barcos}) \bowtie \text{Reservas})) \\ & \rho(\text{Verdetemp}, \pi_{\text{idm}}(\sigma_{\text{color}='verde'}(\text{Barcos}) \bowtie \text{Reservas})) \\ & \pi_{\text{nombrem}}((\text{Rojotemp} \cup \text{Verdetemp}) \bowtie \text{Marineros}) \end{aligned}$$

En la formulación de las consultas C5 y C6 el hecho de que *idm* (el campo sobre el que se calculan la unión o la intersección) sea clave de Marineros es muy importante. Considérese el siguiente intento de responder a la Consulta C6:

$$\begin{aligned} & \rho(\text{Rojotemp}, \pi_{\text{nombrem}}(\sigma_{\text{color}='rojo'}(\text{Barcos}) \bowtie \text{Reservas} \bowtie \text{Marineros})) \\ & \rho(\text{Verdetemp}, \pi_{\text{nombrem}}(\sigma_{\text{color}='verde'}(\text{Barcos}) \bowtie \text{Reservas} \bowtie \text{Marineros})) \\ & \text{Rojotemp} \cap \text{Verdetemp} \end{aligned}$$

Este intento es incorrecto por una razón bastante sutil. Dos marineros diferentes con el mismo nombre, como Horacio en los ejemplares de nuestro ejemplo, pueden haber reservado barcos rojos y verdes, respectivamente. En este caso, el nombre Horacio (incorrectamente) se incluye en la respuesta aunque ninguna persona llamada Horacio ha reservado tanto barcos rojos como barcos verdes. La causa de este error es que *nombrem* se emplea para identificar a los marineros (cuando se lleva a cabo la intersección) en esta versión de la consulta, pero *nombrem* no es una clave.

(C7) Averiguar el nombre de los marineros que han reservado, como mínimo, dos barcos.

$$\begin{aligned} & \rho(Reservas, \pi_{idm, nombrem, idb}(Marineros \bowtie Reservas)) \\ & \rho(Paresreservas(1 \rightarrow idm1, 2 \rightarrow nombrem1, 3 \rightarrow idb1, 4 \rightarrow idm2, \\ & \quad 5 \rightarrow nombrem2, 6 \rightarrow idb2), Reservas \times Reservas) \\ & \pi_{nombrem1}(\sigma_{(idm1=idm2) \wedge (idb1 \neq idb2)}(Paresreservas)) \end{aligned}$$

En primer lugar, se calculan las tuplas de la forma $\langle idm, nombrem, idb \rangle$, donde el marinero *idm* ha hecho una reserva del bote *idb*; este conjunto de tuplas es la relación temporal Reservas. A continuación se buscan todos los pares de tuplas de Reservas en los que el mismo marinero ha hecho las dos reservas y los barcos implicados son diferentes. He aquí la idea central: Para mostrar que un marinero ha reservado dos barcos hay que encontrar dos tuplas de Reservas que impliquen al mismo marinero pero a barcos distintos. En los ejemplares *B1*, *R2* y *M3* cada uno de los marineros con *idms* 22, 31 y 64 ha reservado, como mínimo, dos barcos. Finalmente, se proyectan los nombres de estos marineros para obtener la respuesta, que contiene los nombres Domínguez, Horacio y López.

Obsérvese que se ha incluido *idm* en Reservas, ya que es el campo de la clave que identifica a los marineros y necesitamos que compruebe que dos tuplas cualesquiera de Reservas implican al mismo marinero. Como se indicó en el ejemplo anterior, no se puede emplear *nombrem* con ese fin.

(C8) Averiguar el *idm* de los marineros con edad superior a 20 que no hayan reservado ningún barco rojo.

$$\begin{aligned} & \pi_{idm}(\sigma_{edad > 20}(Marineros)) - \\ & \pi_{idm}(\sigma_{color = 'rojo'}(Barcos) \bowtie Reservas \bowtie Marineros) \end{aligned}$$

Esta consulta ilustra el empleo del operador diferencia de conjuntos. Una vez más se aprovecha el hecho de que *idm* es la clave de Marineros. En primer lugar se identifican los marineros mayores de veinte años (en los ejemplares *B1*, *R2* y *M3*, *idms* 22, 29, 31, 32, 58, 64, 74, 85 y 95) y luego se descartan los que han reservado barcos rojos (*idms* 22, 31 y 64) para obtener la respuesta (*idms* 29, 32, 58, 74, 85 y 95). Si se desea calcular el nombre de estos marineros hay que calcular primero sus *idms* (como ya se ha visto) y luego reunirlos con Marineros y proyectar los valores de *nombrem*.

(C9) Averiguar el nombre de los marineros que han reservado todos los barcos.

El empleo de la palabra *todos* (o *cada*) es una buena indicación de que se podría aplicar la operación división:

$$\begin{aligned} & \rho(Idmstemp, \pi_{idm, idb}(Reservas) / \pi_{idb}(Barcos)) \\ & \pi_{nombrem}(Idmstemp \bowtie Marineros) \end{aligned}$$

La relación intermedia *Idmstemp* se define mediante la división y calcula el conjunto de *idms* de los marineros que han reservado todos los barcos (en los ejemplares *B1*, *R2* y *M3*, esto es, sólo *idm* 22). Obsérvese cómo se han definido las dos relaciones a las que se aplica el operador división (/) —la primera relación tiene el esquema (*idm*, *edb*) y la segunda tiene el esquema (*edb*)—. La división devuelve luego todos los *idms* tales que hay una tupla $\langle idm, idb \rangle$ en la primera relación para cada *edb* de la segunda. La reunión de *Idmstemp* con *Marineros* es necesaria para asociar los nombres con los *idms* seleccionados; para el marinero 22 el nombre es Domínguez.

(C10) *Averiguar el nombre de los marineros que han reservado todos los barcos llamados Intrépido.*

$$\begin{aligned} &\rho(\text{Idmstemp}, \pi_{idm,idb}(\text{Reservas}) / \pi_{edb}(\sigma_{\text{nombreb}=\text{'Intrépido'}}(\text{Barcos}))) \\ &\pi_{nombrem}(\text{Idmstemp} \bowtie \text{Marineros}) \end{aligned}$$

La única diferencia con respecto a la consulta anterior es que ahora se aplica una selección a *Barcos*, para garantizar que sólo se calculan las *idbs* de los barcos llamados *Intrépido* para definir el segundo argumento del operador división. En los ejemplares *B1*, *R2* y *M3*, *Idmstemp* contiene los *idms* 22 y 64, y la respuesta contiene sus nombres, Domínguez y Horacio.

4.3 CÁLCULO RELACIONAL

El cálculo relacional es una alternativa al álgebra relacional. A diferencia del álgebra, que es procedimental, el cálculo es no procedimental, o *declarativo*, en el sentido de que permite describir el conjunto de respuestas sin necesidad de ser explícitos acerca del modo en que se deben calcular. El cálculo relacional ha tenido gran influencia en el diseño de lenguajes de consulta comerciales como SQL y, especialmente, Query-by-Example (QBE).

La variedad del cálculo que se presenta en detalle en este libro se denomina **cálculo relacional de tuplas (CRT)**. Las variables del CRT toman las tuplas como valores. En otra variedad, denominada **cálculo relacional de dominios (CRD)**, las variables adoptan los valores de los campos. El CRT ha tenido cierta influencia en SQL, mientras que el CRD ha influido mucho en QBE. Se tratará el CRD en el Apartado 4.3.2².

4.3.1 Cálculo relacional de tuplas

Una **variable tupla** es una variable que adopta como valores las tuplas de una relación concreta. Es decir, cada valor asignado a una variable tupla dada tiene el mismo número y tipo de campos. Una consulta del cálculo relacional de tuplas tiene la forma $\{ T \mid p(T) \}$, donde *T* es una variable tupla y $p(T)$ denota una *fórmula* que describe *T*; en breve se definirán rigurosamente las fórmulas y las consultas. El resultado de esta consulta es el conjunto de todas las tuplas *t* para las que la fórmula $p(T)$ se evalúa como **verdadera** con $T = t$. El lenguaje para escribir las fórmulas $p(T)$ se halla, por tanto, en el corazón del CRT y, básicamente, es un mero subconjunto de la *lógica de primer orden*. Como ejemplo sencillo, considérese la consulta siguiente.

²Se hace referencia al material sobre el CRD en el capítulo (en Internet) sobre QBE; con la excepción de ese capítulo, el material sobre el CRD y el CRT se puede omitir sin pérdida de continuidad.

(C11) Averiguar todos los marineros con categoría superior a 7.

$$\{M \mid M \in \text{Marineros} \wedge M.\text{categoría} > 7\}$$

Cuando se evalúa esta consulta para algún ejemplar de la relación Marineros, la variable tupla M se instancia sucesivamente con cada tupla, y se aplica la prueba $M.\text{categoría} > 7$. La respuesta contiene aquellos ejemplares de M que superan esta prueba. Para el ejemplar $M3$ de Marineros la respuesta contiene las tuplas de Marineros con *idm* 31, 32, 58, 71 y 74.

Sintaxis de las consultas del CRT

Ahora se definirán formalmente estos conceptos, comenzando con el de fórmula. Sea Rel el nombre de una relación, R y S variables tuplas, a un atributo de R y b un atributo de S . Supongamos que op denota un operador del conjunto $\{<, >, =, \leq, \geq, \neq\}$. Una **fórmula atómica** puede ser:

- $R \in \text{Rel}$
- $R.a \text{ op } S.b$
- $R.a \text{ op constante o constante op } R.a$

Se define recursivamente **fórmula** como una de las posibilidades siguientes, donde p y q son ellas mismas fórmulas y $p(R)$ denota una fórmula en la que aparece la variable R :

- cualquier fórmula atómica
- $\neg p$, $p \wedge q$, $p \vee q$ o $p \Rightarrow q$
- $\exists R(p(R))$, donde R es una variable tupla
- $\forall R(p(R))$, donde R es una variable tupla

En las dos últimas cláusulas se dice que los **cuantificadores** \exists y \forall **ligan** la variable R . Se dice que una variable es **libre** en una fórmula o *subfórmula* (una fórmula contenida en una fórmula mayor) si la (sub)fórmula no contiene ninguna aparición de algún cuantificador que la ligue³.

Obsérvese que cada variable de las fórmulas del CRT aparece en una subfórmula atómica, y todos los esquemas de relación especifican un dominio para cada campo; esta observación garantiza que cada variable de una fórmula del CRT tenga un dominio bien definido del cual extraer los valores de la variable. Es decir, cada variable tiene un *tipo* bien definido, en el sentido de los lenguajes de programación. Informalmente, la fórmula atómica $R \in \text{Rel}$ da a R el tipo de tuplas de Rel , y comparaciones como $R.a \text{ op } S.b$ y $R.a \text{ op constante}$ inducen restricciones de tipos para el campo $R.a$. Si una variable R no aparece en una fórmula atómica de la forma $R \in \text{Rel}$ (es decir, sólo aparece en fórmulas atómicas que son comparaciones), se

³Se hace la suposición de que cada variable de la fórmula es libre o está ligada debido a exactamente un cuantificador. Esto se hace para evitar la preocupación por detalles como los cuantificadores anidados que ligan algunas apariciones de las variables, pero no todas.

deduce el convenio de que el tipo de R es una tupla cuyos campos incluyen todos los campos de R que aparecen en la fórmula (y sólo éstos).

No se definen formalmente los tipos de variable, pero el tipo de cada variable estará claro en la mayor parte de los casos, y lo que hay que destacar es que las comparaciones de valores con tipos diferentes deben fallar siempre. (En discusiones sobre el cálculo relacional se suele hacer la suposición simplificadora de que hay un solo dominio de constantes y que es el dominio asociado con cada campo de cada relación.)

Se define una **consulta del CRT** como una expresión de la forma $\{T \mid p(T)\}$, donde T es la única variable libre de la fórmula p .

Semántica de las consultas del CRT

¿Qué significan las consultas del CRT? Más exactamente, ¿cuál es el conjunto de tuplas respuesta para una consulta del CRT dada? La **respuesta** a una consulta del CRT $\{T \mid p(T)\}$, como ya se ha indicado, es el conjunto de todas las tuplas t para las que la fórmula $p(T)$ se evalúa como **verdadera** con la variable T asignada al valor tupla t . Para completar esta definición hay que indicar las asignaciones de valores tuplas a las variables libres de una fórmula hacen que esa fórmula se evalúe como **verdadera**.

Cada consulta se evalúa sobre un ejemplar dado de la base de datos. Supongamos que cada variable libre de la fórmula F está ligada a un valor tupla. Para la asignación dada de tuplas a las variables, con respecto al ejemplar de la base de datos dada, F se evalúa como (o, sencillamente, “es”) **verdadera** si se cumple alguna de las siguientes condiciones:

- F es una fórmula atómica $R \in Rel$ y a R se le asigna una tupla del ejemplar de la relación Rel .
- F es una comparación $R.a \text{ op } S.b$, $R.a \text{ op } constante$ o $constante \text{ op } R.a$ y las tuplas asignadas a R y a S tienen los valores de campo $R.a$ y $S.b$, que hacen que la comparación sea **verdadera**.
- F es de la forma $\neg p$ y p no es **verdadera**, o de la forma $p \wedge q$, y tanto p como q son **verdaderas**, o de la forma $p \vee q$ y una de ellas es **verdadera**, o de la forma $p \Rightarrow q$ y q es **verdadera** siempre que⁴ p es **verdadera**.
- F es de la forma $\exists R(p(R))$ y hay alguna asignación de tuplas a las variables libres de $p(R)$, incluida la variable R ⁵, que hace que la fórmula $p(R)$ sea **verdadera**.
- F es de la forma $\forall R(p(R))$ y hay alguna asignación de tuplas a las variables libres de $p(R)$ que hace que la fórmula $p(R)$ sea **verdadera** independientemente de la tupla que se asigne a R .

Ejemplos de consultas del CRT

Ahora se ilustrará el cálculo mediante varios ejemplos usando los ejemplares $B1$ de Barcos, $R2$ de Reservas y $M3$ de Marineros que aparecen en las Figuras 4.15, 4.16 y 4.17. Se emplearán

⁴ Siempre se debe entender más exactamente como “para todas las asignaciones de tuplas a las variables libres”.

⁵ Obsérvese que algunas de las variables libres de $p(R)$ (por ejemplo, la propia variable R) puede estar ligada en F .

paréntesis donde hagan falta para evitar cualquier ambigüedad en las fórmulas. A menudo la fórmula $p(R)$ incluye una condición $R \in Rel$ y el significado de las expresiones *alguna tupla de R* y *para todas las tuplas de R* resulta intuitivo. Se empleará la notación $\exists R \in Rel(p(R))$ para $\exists R(R \in Rel \wedge p(R))$. De manera parecida, se emplea la notación $\forall R \in Rel(p(R))$ para $\forall R(R \in Rel \Rightarrow p(R))$.

(C12) Averiguar el nombre y la edad de los marineros de categoría superior a 7.

$$\{P \mid \exists M \in Marineros(M.\text{categoría} > 7 \wedge P.\text{nombre} = M.\text{nombrem} \wedge P.\text{edad} = M.\text{edad})\}$$

Esta consulta ilustra un convenio muy útil: se considera que P es una variable tupla exactamente con dos campos, que se denominan *nombre* y *edad*, ya que son los únicos campos de P mencionados y P no recorre ninguna de las relaciones de la consulta; es decir, no hay ninguna subfórmula de la forma $P \in Nombrel$. El resultado de esta consulta es una relación con dos campos, *nombre* y *edad*. Las fórmulas atómicas $P.\text{nombre} = M.\text{nombrem}$ y $P.\text{edad} = M.\text{edad}$ dan valores a los campos de la tupla respuesta P . Para los ejemplares $B1$, $R2$ y $M3$ la respuesta es el conjunto de tuplas $\langle \text{López}, 55, 5 \rangle$, $\langle \text{Arturo}, 25, 5 \rangle$, $\langle \text{Rubio}, 35, 0 \rangle$, $\langle \text{Zuazo}, 16, 0 \rangle$ y $\langle \text{Horacio}, 35, 0 \rangle$.

(C13) Averiguar el nombre del marinero, el identificador del barco y la fecha de reserva de cada reserva.

$$\{P \mid \exists R \in Reservas \ \exists M \in Marineros$$

$$(R.\text{idm} = M.\text{idm} \wedge P.\text{idb} = R.\text{idb} \wedge P.\text{fecha} = R.\text{fecha} \wedge P.\text{nombrem} = M.\text{nombrem})\}$$

Para cada tupla de Reservas se busca una tupla de Marineros con la misma *idm*. Dado un par de esas tuplas, se crea la tupla respuesta P con los campos *nombrem*, *idb* y *fecha* copiando los campos correspondientes de esas dos tuplas. Esta consulta ilustra la manera en que se pueden combinar valores de relaciones diferentes en cada tupla respuesta. La respuesta a esta consulta para los ejemplares $B1$, $R2$ y $M3$ puede verse en la Figura 4.20.

<i>nombrem</i>	<i>idb</i>	<i>fecha</i>
Domínguez	101	10/10/98
Domínguez	102	10/10/98
Domínguez	103	10/8/98
Domínguez	104	10/7/98
López	102	11/10/98
López	103	11/6/98
López	104	11/12/98
Horacio	101	9/5/98
Horacio	102	9/8/98
Horacio	103	9/8/98

Figura 4.20 Respuesta a la consulta C13

(C1) Averiguar el nombre de los marineros que han reservado el barco 103.

$$\{P \mid \exists M \in Marineros \ \exists R \in Reservas(R.\text{idm} = M.\text{idm} \wedge R.\text{idb} = 103 \wedge P.\text{nombrem} = M.\text{nombrem})\}$$

Esta consulta se puede leer de la manera siguiente: “Recuperar todas las tuplas de Marineros para las cuales existe una tupla de Reservas que tiene el mismo valor del campo *idm* y con *idb* = 103.” Es decir, por cada tupla de Marineros hay que buscar una tupla de Reservas que muestre que ese marinero ha reservado el barco 103. La tupla respuesta *P* sólo contiene un campo, *nombrem*.

(C2) Averiguar el nombre de los marineros que han reservado barcos rojos.

$$\{P \mid \exists M \in \text{Marineros} \ \exists R \in \text{Reservas} (R.idm = M.idm \wedge P.nombrem = M.nombrem \wedge \exists B \in \text{Barcos} (B.idb = R.idb \wedge B.color = 'rojo'))\}$$

Esta consulta se puede leer de la manera siguiente: “Recuperar todas las tuplas *M* de Marineros para las cuales existan tuplas *R* de Reservas y *B* de Barcos tales que *M.idm* = *R.idm*, *R.idb* = *B.idb*, y *B.color* = ‘rojo’.” Otra manera de escribir esta consulta, que se corresponde mejor con esta lectura, es la siguiente:

$$\{P \mid \exists M \in \text{Marineros} \ \exists R \in \text{Reservas} \ \exists B \in \text{Barcos} (R.idm = M.idm \wedge B.idb = R.idb \wedge B.color = 'rojo' \wedge P.nombrem = M.nombrem)\}$$

(C7) Averiguar el nombre de los marineros que han reservado, como mínimo, dos barcos.

$$\begin{aligned} & \{P \mid \exists M \in \text{Marineros} \ \exists R1 \in \text{Reservas} \ \exists R2 \in \text{Reservas} \\ & (M.idm = R1.idm \wedge R1.idm = R2.idm \wedge R1.idb \neq R2.idb \\ & \wedge P.nombrem = M.nombrem)\} \end{aligned}$$

Compárese esta consulta con la versión del álgebra relacional y véase cómo la versión del CRT es mucho más sencilla. En parte, esta diferencia se debe al engorroso renombramiento de los campos de la versión del álgebra relacional, pero la versión del CRT es realmente más sencilla.

(C9) Averiguar el nombre de los marineros que han reservado todos los barcos.

$$\{P \mid \exists M \in \text{Marineros} \ \forall B \in \text{Barcos} (\exists R \in \text{Reservas} (M.idm = R.idm \wedge R.idb = B.idb \wedge P.nombrem = M.nombrem))\}$$

Esta consulta se expresó en el álgebra relacional mediante el operador división. Obsérvese lo fácilmente que se expresa en el cálculo relacional. La consulta del CRT refleja directamente la manera en que se expresaría la consulta en castellano: “Averiguar los marineros de *M* tales que para todos los barcos de *B* hay una tupla de Reservas que muestra que el marinero *M* ha reservado el barco *B*.”

(C14) Averiguar los marineros que han reservado todos los barcos rojos.

$$\{M \mid M \in \text{Marineros} \wedge \forall B \in \text{Barcos} (B.color = 'rojo' \Rightarrow (\exists R \in \text{Reservas} (M.idm = R.idm \wedge R.idb = B.idb)))\}$$

Esta consulta puede leerse de la manera siguiente: “Para cada candidato (marinero), si el barco es rojo, el marinero debe haberlo reservado.” Es decir, para cada marinero candidato, que un barco sea rojo debe implicar que el marinero lo ha reservado. Obsérvese que, dado que

se puede devolver toda una tupla de Marineros como respuesta en lugar de sólo el nombre del marinero, se evitó introducir una nueva variable libre (por ejemplo, la variable P del ejemplo anterior) para que guardara los valores respuesta. Para los ejemplares $B1$, $R2$ y $M3$ la respuesta contiene las tuplas de Marineros con $idms$ 22 y 31.

Esta consulta se puede escribir sin emplear la implicación, observando que las expresiones de la forma $p \Rightarrow q$ son equivalentes lógicamente a $\neg p \vee q$:

$$\{M \mid M \in \text{Marineros} \wedge \forall B \in \text{Barcos} \\ (B.\text{color} \neq 'rojo' \vee (\exists R \in \text{Reservas}(M.idm = R.idm \wedge R.idb = B.idb)))\}$$

Esta consulta se debe leer de la manera siguiente: “Averiguar los marineros M tales que, para todos los barcos B , o bien el barco no es rojo o alguna tupla de Reservas muestra que el marinero M ha reservado el barco B .”

4.3.2 Cálculo relacional de dominios

Una **variable de dominio** es una variable que recorre los valores del dominio de algún atributo (por ejemplo, se le puede asignar un entero a la variable si aparece en un atributo cuyo dominio sea el conjunto de enteros). Las consultas del CRD tienen la forma $\{\langle x_1, x_2, \dots, x_n \rangle \mid p(\langle x_1, x_2, \dots, x_n \rangle)\}$, donde cada x_i es una *variable de dominio* o una constante, y $p(\langle x_1, x_2, \dots, x_n \rangle)$ denota una **fórmula del CRD** cuyas únicas variables libres son las que se hallan entre las x_i , $1 \leq i \leq n$. El resultado de esta consulta es el conjunto de todas las tuplas $\langle x_1, x_2, \dots, x_n \rangle$ para las que la fórmula se evalúa como **verdadera**.

Las fórmulas del CRD se definen de manera muy parecida a las fórmulas del CRT. La principal diferencia es que ahora las variables son variables de dominio. Supongamos que op denota un operador del conjunto $\{<, >, =, \leq, \geq, \neq\}$ y que X y Y son variables de dominio. Una **fórmula atómica** en el CRD es una de las siguientes:

- $\langle x_1, x_2, \dots, x_n \rangle \in Rel$, donde Rel es una relación con n atributos; cada x_i , $1 \leq i \leq n$ es una variable o una constante
- $X \text{ op } Y$
- $X \text{ op constante o constante op } X$

Las **fórmulas** se definen recursivamente como una de las siguientes posibilidades, donde p y q son fórmulas y $p(X)$ denota una fórmula en la que aparece la variable X :

- cualquier fórmula atómica
- $\neg p$, $p \wedge q$, $p \vee q$ o $p \Rightarrow q$
- $\exists X(p(X))$, donde X es una variable de dominio
- $\forall X(p(X))$, donde X es una variable de dominio

Se invita al lector a comparar esta definición con la de las fórmulas del CRT y ver lo bien que se corresponden las dos definiciones. No se definirá formalmente la semántica de las fórmulas del CRD; se deja como ejercicio para el lector.

Ejemplos de consultas del CRD

Ahora se ilustrará el CRD mediante varios ejemplos. Se invita al lector a compararlos con sus versiones para el CRT.

(C11) Averiguar todos los marineros con categoría superior a 7.

$$\{\langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \text{Marineros} \wedge T > 7\}$$

Esto se diferencia de la versión para el CRT en que se da a cada atributo un nombre (de variable). La condición $\langle I, N, T, A \rangle \in \text{Marineros}$ garantiza que las variables de dominio I , N , T y A estén restringidas a los campos de la *misma* tupla. En comparación con la consulta del CRT, se puede decir $T > 7$ en lugar de $M.\text{categoría} > 7$, pero hay que especificar en el resultado la tupla $\langle I, N, T, A \rangle$, en lugar de sólo M .

(C1) Averiguar el nombre de los marineros que han reservado el barco 103.

$$\begin{aligned} &\{\langle N \rangle \mid \exists I, T, A (\langle I, N, T, A \rangle \in \text{Marineros} \\ &\quad \wedge \exists Ir, Br, D (\langle Ir, Br, D \rangle \in \text{Reservas} \wedge Ir = I \wedge Br = 103))\} \end{aligned}$$

Obsérvese que sólo se conserva en la respuesta el campo *nombrem* y que sólo N es una variable libre. Se emplea la notación $\exists Ir, Br, D(\dots)$ como abreviatura de $\exists Ir (\exists Br (\exists D(\dots)))$. Muy a menudo todas las variables cuantificadas aparecen en una sola relación, como en este ejemplo. Una notación todavía más compacta en este caso es $\exists \langle Ir, Br, D \rangle \in \text{Reservas}$. Con esta notación, que utilizamos de aquí en adelante, la consulta queda de la manera siguiente:

$$\begin{aligned} &\{\langle N \rangle \mid \exists I, T, A (\langle I, N, T, A \rangle \in \text{Marineros} \\ &\quad \wedge \exists \langle Ir, Br, D \rangle \in \text{Reservas} (Ir = I \wedge Br = 103))\} \end{aligned}$$

La comparación con la fórmula correspondiente del CRT ahora debe ser sencilla. Esta consulta también se puede escribir de la manera siguiente; obsérvese la repetición de la variable I y el empleo de la constante 103:

$$\begin{aligned} &\{\langle N \rangle \mid \exists I, T, A (\langle I, N, T, A \rangle \in \text{Marineros} \\ &\quad \wedge \exists D (\langle I, 103, D \rangle \in \text{Reservas}))\} \end{aligned}$$

(C2) Averiguar el nombre de los marineros que han reservado barcos rojos.

$$\begin{aligned} &\{\langle N \rangle \mid \exists I, T, A (\langle I, N, T, A \rangle \in \text{Marineros} \\ &\quad \wedge \exists \langle Ir, Br, D \rangle \in \text{Reservas} \wedge \exists \langle Br, NB, 'rojo' \rangle \in \text{Barcos})\} \end{aligned}$$

(C7) Averiguar el nombre de los marineros que han reservado, como mínimo, dos barcos.

$$\begin{aligned} &\{\langle N \rangle \mid \exists I, T, A (\langle I, N, T, A \rangle \in \text{Marineros} \wedge \\ &\quad \exists Br1, Br2, D1, D2 (\langle I, Br1, D1 \rangle \in \text{Reservas} \\ &\quad \wedge \langle I, Br2, D2 \rangle \in \text{Reservas} \wedge Br1 \neq Br2))\} \end{aligned}$$

Obsérvese cómo el empleo reiterado de la variable I garantiza que el mismo marinero haya reservado los dos barcos en cuestión.

(C9) Averiguar el nombre de los marineros que han reservado todos los barcos.

$$\{\langle N \rangle \mid \exists I, T, A (\langle I, N, T, A \rangle \in \text{Marineros} \wedge \forall B, NB, C (\neg(\langle B, NB, C \rangle \in \text{Barcos}) \vee (\exists \langle Ir, Br, D \rangle \in \text{Reservas} (I = Ir \wedge Br = B))))\}$$

Esta consulta se puede leer de la manera siguiente: “Averiguar todos los valores de N tales que alguna tupla $\langle I, N, T, A \rangle$ de Marineros satisface la condición siguiente: para cada $\langle B, NB, C \rangle$, o bien no se trata de una tupla de Barcos o hay alguna tupla $\langle Ir, Br, D \rangle$ de Reservas que prueba que el Marinero I ha reservado el barco B . ” El cuantificador \forall permite que las variables de dominio B , NB y C recorran todos los valores de los dominios de sus atributos respectivos, y la estructura “ $\neg(\langle B, NB, C \rangle \in \text{Barcos}) \vee$ ” es necesaria para restringir la atención a aquellos valores que aparecen en las tuplas de Barcos. Esta estructura es frecuente en las fórmulas del CRD, y la notación $\forall \langle B, NB, C \rangle \in \text{Barcos}$ se puede utilizar como abreviatura en su lugar. Esto es parecido a la notación introducida anteriormente para \exists . Con esta notación la consulta se escribiría de la manera siguiente:

$$\{\langle N \rangle \mid \exists I, T, A (\langle I, N, T, A \rangle \in \text{Marineros} \wedge \forall \langle B, NB, C \rangle \in \text{Barcos} (\exists \langle Ir, Br, D \rangle \in \text{Reservas} (I = Ir \wedge Br = B)))\}$$

(C14) Averiguar los marineros que han reservado todos los barcos rojos.

$$\{\langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \text{Marineros} \wedge \forall \langle B, NB, C \rangle \in \text{Barcos} (C = 'rojo' \Rightarrow \exists \langle Ir, Br, D \rangle \in \text{Reservas} (I = Ir \wedge Br = B))\}$$

En este caso se buscan todos los marineros tales que, para cada barco rojo, hay una tupla de Reservas que muestra que ese marinero lo ha reservado.

4.4 POTENCIA EXPRESIVA DEL ÁLGEBRA Y DEL CÁLCULO

Se han presentado dos lenguajes formales de consulta para el modelo relacional. ¿Son equivalentes en potencia? ¿Pueden expresarse también en el cálculo relacional todas las consultas que pueden expresarse en el álgebra relacional? La respuesta es sí, pueden. ¿Pueden expresarse también en el álgebra relacional todas las consultas que pueden expresarse en el cálculo relacional? Antes de contestar esta pregunta, considérese un problema importante del cálculo tal y como se ha presentado aquí.

Considérese la consulta $\{M \mid \neg(M \in \text{Marineros})\}$. Esta consulta es sintácticamente correcta. Sin embargo, pide todas las tuplas M tales que M no se halle en (el ejemplar dado de) Marineros. El conjunto de esas tuplas M es, evidentemente, infinito, en el contexto de los dominios infinitos como el conjunto de todos los enteros. Este ejemplo sencillo ilustra una consulta *insegura*. Resulta deseable restringir el cálculo relacional para impedir las consultas inseguras.

Ahora se esbozará la manera en que se restringen las consultas para que sean seguras. Considérese el conjunto E de ejemplares de las relaciones, con un ejemplar por relación que

aparezca en la consulta C . Sea $\text{Dom}(C, E)$ el conjunto de todas las constantes que aparecen en esos ejemplares de las relaciones E o en la formulación de la propia consulta C . Dado que sólo se permiten ejemplares E finitos, $\text{Dom}(C, E)$ también es finito.

Para que una fórmula del cálculo C sea considerada segura, como mínimo hay que garantizar que, para cualquier E dado, el conjunto de respuestas de C sólo contiene valores de $\text{Dom}(C, E)$. Aunque es evidente que esta restricción se necesita, no es suficiente. No sólo se desea que el conjunto de respuestas esté compuesto de constantes de $\text{Dom}(C, E)$, se desea *calcular* el conjunto de respuestas examinando únicamente las tuplas que contienen constantes de $\text{Dom}(C, E)$. Esta necesidad provoca un problema sutil asociado al empleo de los cuantificadores \forall y \exists : dada una fórmula del CRT de la forma $\exists R(p(R))$, se desea hallar todos los valores de la variable R que hace que esta fórmula sea **verdadera** comprobando sólo las tuplas que contienen constantes de $\text{Dom}(C, E)$. De manera parecida, dada una fórmula del CRT de la forma $\forall R(p(R))$, se desea hallar cualquier valor de la variable R que haga que esta fórmula sea **falsa** comprobando sólo las tuplas que contienen constantes de $\text{Dom}(C, E)$.

Por tanto, una fórmula del CRT *segura* C se define como una fórmula tal que:

1. Para cualquier E dado, el conjunto de respuestas de C sólo contiene valores que se hallan en $\text{Dom}(C, E)$.
2. Para cada subexpresión de la forma $\exists R(p(R))$ de C , si una tupla r (asignada a la variable R) hace que la fórmula sea **verdadera**, entonces r sólo contiene constantes de $\text{Dom}(C, E)$.
3. Para cada subexpresión de la forma $\forall R(p(R))$ de C , si una tupla r (asignada a la variable R) contiene una constante que no se halla en $\text{Dom}(C, E)$, entonces r debe hacer que la fórmula sea **verdadera**.

Obsérvese que esta definición no es *constructiva*, es decir, no indica la manera de comprobar si una consulta dada es segura.

La consulta $C = \{M \mid \neg(M \in \text{Marineros})\}$ es insegura de acuerdo con esta definición. $\text{Dom}(C, E)$ es el conjunto de todos los valores que aparecen en (el ejemplar E de) Marineros. Considérese el ejemplar $M1$ de la Figura 4.1. La respuesta a esta consulta, evidentemente, incluye valores que no aparecen en $\text{Dom}(C, M1)$.

Volviendo al asunto de la expresividad, se puede probar que todas las consultas que pueden expresarse mediante consultas *seguras* del cálculo relacional se pueden expresar también como consultas del álgebra relacional. La potencia expresiva del álgebra relacional se suele emplear como medida de la potencia de los lenguajes de consulta de las bases de datos relacionales. Si un lenguaje de consulta puede expresar todas las consultas que se pueden expresar en el álgebra relacional, se dice que es **relacionalmente completo**. Se espera que los lenguajes de consulta prácticos sean relationalmente completos; además, los lenguajes comerciales de consulta suelen soportar características que permiten expresar algunas consultas que no pueden expresarse en el álgebra relacional.

4.5 PREGUNTAS DE REPASO

Las respuestas a las preguntas de repaso se pueden encontrar en los apartados indicados.

- ¿Cuáles son los datos de las consultas relacionales? ¿Cuál es el resultado de la evaluación de una consulta? (**Apartado 4.1**)

- Los sistemas de bases de datos emplean alguna variante del álgebra relacional para representar los planes de evaluación de las consultas. Explíquese el motivo de que el álgebra resulte adecuada para esta finalidad. (**Apartado 4.2**)
- Describase el operador selección. ¿Qué se puede decir sobre la cardinalidad de las tablas de entrada y de salida de este operador? (Es decir, si la entrada tiene k tuplas, ¿qué se puede decir de la salida?) Describase el operador proyección. ¿Qué se puede decir sobre la cardinalidad de las tablas de entrada y de salida de este operador? (**Apartado 4.2.1**)
- Describanse las operaciones con conjuntos del álgebra relacional, incluidas la unión (\cup), la intersección (\cap), la diferencia de conjuntos ($-$) y el producto cartesiano (\times). Para cada una de ellas, ¿qué se puede decir sobre la cardinalidad de sus tablas de entrada y de salida? (**Apartado 4.2.2**)
- Explíquese la manera en que se emplea el operador renombramiento. ¿Es necesario? Es decir, si no se permitiera este operador, ¿hay alguna consulta que ya no se pudiese expresar en el álgebra? (**Apartado 4.2.3**)
- Defínanse todas las variaciones de la operación reunión. ¿Por qué se presta atención especial a la operación reunión? ¿No se pueden expresar todas las operaciones reunión en términos del producto cartesiano, la selección y la proyección? (**Apartado 4.2.4**)
- Defínase la operación división en términos de las operaciones básicas del álgebra relacional. Describase una consulta típica que requiera la división. A diferencia de la reunión, el operador división no recibe ningún tratamiento especial en los sistemas de bases de datos. Explíquese el motivo. (**Apartado 4.2.5**)
- Se dice que el cálculo relacional es un lenguaje *declarativo*, a diferencia del álgebra, que es un lenguaje *procedimental*. Explíquese la diferencia. (**Apartado 4.3**)
- ¿Cómo “describen” las consultas del cálculo relacional las tuplas del resultado? Examíñese el subconjunto de la lógica de predicados de primer orden empleado en el cálculo relacional de tuplas, con especial atención a los cuantificadores universal y existencial, las variables ligadas y libres, y las restricciones sobre las fórmulas de consulta. (**Apartado 4.3.1**)
- ¿Cuál es la diferencia entre el cálculo relacional de tuplas y el cálculo relacional de dominios? (**Apartado 4.3.2**)
- ¿Qué son las consultas *inseguras* del cálculo? ¿Por qué es importante evitar esas consultas? (**Apartado 4.4**)
- Se dice que el álgebra relacional y el cálculo relacional son equivalentes en potencia expresiva. Explíquese lo que esto significa y cómo se relaciona con el concepto de *completitud relacional*. (**Apartado 4.4**)

EJERCICIOS

Ejercicio 4.1 Explíquese la afirmación de que los operadores del álgebra relacional pueden *componerse*. ¿Por qué es importante la posibilidad de componer operadores?

Ejercicio 4.2 Dadas dos relaciones $R1$ y $R2$, donde $R1$ contiene $N1$ tuplas, $R2$ contiene $N2$ tuplas y $N2 > N1 > 0$, dízase los tamaños mínimo y máximo posibles (en tuplas) de la relación resultante de cada una de las siguientes expresiones del álgebra relacional. En cada caso, formulense algunas suposiciones sobre los esquemas de $R1$ y de $R2$ necesarias para hacer que la expresión tenga sentido:

- (1) $R1 \cup R2$, (2) $R1 \cap R2$, (3) $R1 - R2$, (4) $R1 \times R2$, (5) $\sigma_{a=5}(R1)$, (6) $\pi_a(R1)$ y (7) $R1/R2$

Ejercicio 4.3 Considérese el esquema siguiente:

Proveedores(idp : integer, $nombrep$: string, $domicilio$: string)

Repuestos(idr : integer, $nombrer$: string, $color$: string)

Catálogo(idp : integer, idr : integer, $coste$: real)

Los campos de la clave están subrayados, y el dominio de cada campo se relaciona tras el nombre correspondiente. Por tanto, idp es la clave de Proveedores, idr es la de Repuestos e idp e idr forman conjuntamente la clave de Catálogo. La relación catálogo muestra los precios cobrados por los Proveedores por los repuestos. Escríbanse las consultas siguientes en el álgebra relacional, el cálculo relacional de tuplas y el cálculo relacional de dominios:

1. Averiguar el *nombre* de los proveedores que suministran algún repuesto rojo.
2. Averiguar el *idp* de los proveedores que suministran algún repuesto rojo o verde.
3. Averiguar el *idp* de los proveedores que suministran algún repuesto rojo o se hallan en el 221 de la Calle de la Empaquetadora.
4. Averiguar el *idp* de los proveedores que suministran algún repuesto rojo y algún repuesto verde.
5. Averiguar el *idp* de los proveedores que suministran todos los repuestos.
6. Averiguar el *idp* de los proveedores que suministran todos los repuestos rojos.
7. Averiguar el *idp* de los proveedores que suministran todos los repuestos rojos o verdes.
8. Averiguar el *idp* de los proveedores que suministran todos los repuestos rojos o todos los repuestos verdes.
9. Buscar parejas de *idps* tales que el proveedor con el primer *idp* cobre más por algún repuesto que el proveedor con el segundo *idp*.
10. Averiguar el *idr* de los repuestos suministrados por, al menos, dos proveedores diferentes.
11. Averiguar el *idr* de los repuestos más caros suministrados por los proveedores llamados Yelmo Suárez.
12. Averiguar el *idr* de los repuestos suministrados por todos los proveedores por menos de 200 €. (Si algún proveedor no suministra ese repuesto o lo cobra a más de 200 €, no se selecciona.)

Ejercicio 4.4 Considérese el esquema Proveedor-Repuestos-Catálogo de la pregunta anterior. Indíquese lo que calculan las consultas siguientes:

1. $\pi_{nombrep}(\pi_{idp}(\sigma_{color='rojo'}(Repuestos)) \bowtie \sigma_{coste<100}(Catálogo)) \bowtie Proveedores$
2. $\pi_{nombrep}(\pi_{idp}(\sigma_{color='rojo'}(Repuestos)) \bowtie \sigma_{coste<100}(Catálogo) \bowtie Proveedores)$
3. $(\pi_{nombrep}(\sigma_{color='rojo'}(Repuestos)) \bowtie \sigma_{coste<100}(Catálogo) \bowtie Proveedores) \cap (\pi_{nombrep}(\sigma_{color='verde'}(Repuestos)) \bowtie \sigma_{coste<100}(Catálogo) \bowtie Proveedores)$
4. $(\pi_{idp}(\sigma_{color='rojo'}(Repuestos)) \bowtie \sigma_{coste<100}(Catálogo) \bowtie Proveedores) \cap (\pi_{idp}(\sigma_{color='verde'}(Repuestos)) \bowtie \sigma_{coste<100}(Catálogo) \bowtie Proveedores)$

5. $\pi_{nombrep}((\pi_{idp,nombrep}(\sigma_{color='rojo'}(Repuestos) \bowtie \sigma_{coste<100}(Catálogo) \bowtie Proveedores)) \cap (\pi_{idp,nombrep}(\sigma_{color='verde'}(Repuestos) \bowtie \sigma_{coste<100}(Catálogo) \bowtie Proveedores)))$

Ejercicio 4.5 Considérense las siguientes relaciones que contienen información de vuelos de líneas aéreas:

```
Vuelos(nuvu: integer, de: string, a: string,
       distancia: integer, despegue: time, aterrizaje: time)
Avión(ida: integer, nombrea: string, autonomía: integer)
Certificado(ide: integer, ida: integer)
Empleados(ide: integer, nombree: string, sueldo: integer)
```

Obsérvese que la relación Empleados describe a los pilotos y también a otros tipos de empleados; todos los pilotos están certificados para algún tipo de avión (en caso contrario, no se considerarían pilotos) y sólo los pilotos están certificados para volar.

Escríbanse las consultas siguientes en el álgebra relacional, el cálculo relacional de tuplas y el cálculo relacional de dominios. Obsérvese que puede que algunas de estas consultas no se puedan expresar en el álgebra relacional (y, por tanto, tampoco en el cálculo relacional de tuplas ni en el de dominios). Para esas consultas explíquese de manera informal el motivo de que no se puedan expresar. (Véanse los ejercicios al final del Capítulo 5 para encontrar más consultas sobre el esquema de la aerolínea.)

1. Averiguar el *ide* de los pilotos certificados para algún avión de Airbus.
2. Averiguar el *nombre* de los pilotos certificados para algún avión de Airbus.
3. Averiguar el *ida* de todos los aviones que pueden emplearse para vuelos sin escalas entre Barcelona y Medellín.
4. Identificar los vuelos que puede pilotar cada piloto cuyo sueldo es superior a 100.000 €.
5. Averiguar el nombre de los pilotos que pueden operar aviones con una autonomía mayor de 5.500 kilómetros pero que no están certificados para ningún avión de Airbus.
6. Averiguar el *ide* de los empleados que tienen el sueldo más elevado.
7. Averiguar el *ide* de los empleados que tienen el segundo sueldo más elevado.
8. Averiguar el *ide* de los empleados que están certificados para mayor número de aviones.
9. Averiguar el *ide* de los empleados que están certificados exactamente para tres aviones.
10. Averiguar el importe total pagado a los empleados como sueldo.
11. ¿Hay alguna secuencia de vuelos de Maracaibo a Tumbuctú? Se exige que cada vuelo de la secuencia despegue de la ciudad que es el destino del vuelo anterior; el primer vuelo debe salir de Maracaibo, el último vuelo debe llegar a Tumbuctú y no hay ninguna restricción en cuanto al número de vuelos intermedios. La consulta debe determinar si existe alguna secuencia de vuelos de Maracaibo a Tumbuctú para cualquier ejemplar de la relación Vuelos entrante.

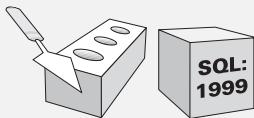
Ejercicio 4.6 ¿Qué es la *completitud relacional*? Si un lenguaje de consulta es relationalmente completo, ¿se puede escribir en ese lenguaje cualquier consulta que se deseé?

Ejercicio 4.7 ¿Qué es una consulta *insegura*? Dese un ejemplo y explíquese el motivo de que sea importante impedir esas consultas.

NOTAS BIBLIOGRÁFICAS

Codd propuso el álgebra relacional en [119], y mostró la equivalencia entre el álgebra relacional y el CRT en [121]. Anteriormente, Kuhns [287] consideró el empleo de la lógica para plantear las consultas. LaCroix

y Pirotte trataron el CRD en [288]. Klug generalizó el álgebra y el cálculo para incluir las operaciones de agregación en [280]. También se discuten extensiones del álgebra y del cálculo para tratar con las funciones de agregación en [349]. Merrett propuso un álgebra relacional ampliado con cuantificadores como “*el número de*” que va más allá de la mera cuantificación universal y existencial [330]. Esos cuantificadores generalizados se estudian en profundidad en [36].



5

SQL: CONSULTAS, RESTRICCIONES Y DISPARADORES

- ➔ ¿Qué incluye el lenguaje SQL? ¿Qué es SQL:1999?
- ➔ ¿Cómo expresan las consultas SQL?
¿Cuál es el significado de las consultas expresadas según la norma de SQL?
- ➔ ¿Cómo aprovecha SQL el álgebra y el cálculo relacionales y cómo los amplía?
- ➔ ¿Qué es la agrupación? ¿Cómo se emplea en las operaciones de agregación?
- ➔ ¿Qué son las consultas anidadas?
- ➔ ¿Qué son los valores nulos (*null*)?
- ➔ ¿Cómo se pueden utilizar las consultas para escribir restricciones de integridad complejas?
- ➔ ¿Qué son los disparadores y por qué resultan útiles? ¿Qué relación tienen con las restricciones de integridad?
- ➡ **Conceptos fundamentales:** consultas SQL, conexión con el álgebra y el cálculo relacionales; características adicionales al álgebra, la cláusula DISTINCT y la semántica multiconjunto, agrupación y agregación; consultas anidadas, correlación; operadores para la comparación de conjuntos; valores nulos (*null*), reuniones externas; restricciones de integridad especificadas mediante consultas; disparadores y bases de datos activas, reglas evento-condición-acción.

¿Qué hombres o dioses son éstos? ¿Qué doncellas reticentes?
¿Qué loco empeño? ¿Qué lucha por escapar?
¿Qué flautas y panderetas? ¿Qué salvaje éxtasis?

— John Keats, *Oda a una urna griega*

El lenguaje estructurado de consulta (Structured Query Language, SQL) es el lenguaje comercial para bases de datos relacionales más utilizado. Se desarrolló originalmente en IBM en los

Conformidad con las normas de SQL. SQL:1999 tiene un conjunto de características denominado Core SQL (SQL Fundamental) que los fabricantes deben implementar para poder afirmar que cumplen la norma SQL:1999. Se estima que todos los principales fabricantes pueden cumplir Core SQL con poco esfuerzo. Gran parte del resto de características se organiza en **paquetes**.

Por ejemplo, los paquetes tratan las siguientes características se tratan mediante (se indican los capítulos correspondientes entre paréntesis): *tratamiento mejorado de la fecha y de la hora, gestión mejorada de la integridad y bases de datos activas* (este capítulo), *interfaces para lenguajes externos* (Capítulo 6), *OLAP* (Capítulo 15) y *características orientadas a objetos* (Capítulo 15). La norma SQL/MM complementa SQL:1999 mediante la definición de paquetes adicionales que soportan la *minería de datos* (Capítulo 17), los *datos espaciales* (no tratado en este texto) y los *documentos de texto* (Capítulo 18). El soporte para los datos y las consultas XML es inminente.

proyectos SEQUEL-XRM y System-R (1974-1977). Casi inmediatamente, otros fabricantes introdujeron productos de SGBD basados en SQL, y ahora es la norma de facto. SQL sigue evolucionando en respuesta a las necesidades cambiantes del área de las bases de datos. La norma ANSI/ISO para SQL que se trata en este libro es SQL:1999. Aunque no todos los productos de SGBD soportan todavía toda la norma SQL:1999, los fabricantes trabajan para conseguir ese objetivo, y la mayor parte de los productos ya soportan las características principales. La norma SQL:1999 es muy similar a la norma anterior, SQL-92, con respecto a las características que se estudian en este capítulo. La presentación que se ofrece es consistente tanto con SQL-92 como con SQL:1999, y los aspectos que no son iguales en las dos versiones de la norma se destacan de manera explícita.

5.1 INTRODUCCIÓN

El lenguaje SQL tiene varios aspectos diferentes.

- **Lenguaje de manipulación de datos (LMD).** Este subconjunto de SQL permite a los usuarios formular consultas e insertar, eliminar y modificar filas. Las consultas son el principal objeto de atención de este capítulo. Las órdenes del LMD para insertar, eliminar y modificar filas se trataron en el Capítulo 3.
- **Lenguaje de definición de datos (LDD).** Este subconjunto de SQL soporta la creación, eliminación y modificación de definiciones de tablas y vistas. Se pueden definir *restricciones de integridad* para las tablas, bien en el momento de crearlas, bien posteriormente. Las características del LDD de SQL se trataron en el Capítulo 3. Aunque la norma no estudia los índices, las implementaciones comerciales ofrecen también órdenes para la creación y eliminación de índices.
- **Disparadores y restricciones de integridad avanzadas.** La nueva norma SQL:1999 incluye soporte para los *disparadores*, que son acciones ejecutadas por el SGBD siempre

que las modificaciones de la base de datos cumplen las condiciones especificadas en el disparador. Los disparadores se tratan en este capítulo. SQL permite el empleo de consultas para definir especificaciones complejas de restricciones de integridad. También se examinan esas restricciones en este capítulo.

- **SQL incorporado y SQL dinámico.** Las características de SQL incorporado permiten llamar al código SQL desde lenguajes anfitriones como C o COBOL. Las características de SQL dinámico permiten que se creen (y ejecuten) consultas en el momento de la ejecución. Estas características se tratan en el Capítulo 6.
- **Ejecución cliente-servidor y acceso a bases de datos remotas.** Estas órdenes controlan el modo en que los programas de aplicación *clientes* pueden conectarse con los *servidores* de bases de datos de SQL o tener acceso a los datos de las bases de datos a través de la red. Estas órdenes se tratan en el Capítulo 7.
- **Gestión de transacciones.** Diversas órdenes permiten que los usuarios controlen de manera explícita aspectos del modo en que se deben ejecutar las transacciones. Estas órdenes se tratan en el Capítulo 14.
- **Seguridad.** SQL ofrece mecanismos para control el acceso de los usuarios a los objetos de datos, como tablas y vistas. Esto se trata en el Capítulo 14.
- **Características avanzadas.** La norma SQL:1999 incluye características orientadas a objetos (Capítulo 15), consultas de apoyo a las decisiones (Capítulo 16) y también aborda áreas emergentes como la minería de datos (Capítulo 17), los datos espaciales y la gestión de datos de texto y de XML (Capítulo 18).

5.1.1 Organización del capítulo

El resto de este capítulo se organiza de la manera siguiente. Las consultas SQL se presentan en el Apartado 5.2 y los operadores para conjuntos de SQL en el Apartado 5.3. Las consultas anidadas, en las que relaciones a las que se hace referencia en la consulta se definen a su vez en la propia consulta, se tratan en el Apartado 5.4. Los operadores de agregación, que permiten escribir consultas SQL que no se pueden expresar en el álgebra relacional, se tratan en el Apartado 5.5. Los valores *null*, que son valores especiales empleados para indicar que el valor de un campo es desconocido o no existe, se estudian en el Apartado 5.6. Las restricciones de integridad complejas que se pueden especificar mediante el LDD de SQL se estudian en el Apartado 5.7, lo que amplía el estudio del LDD de SQL del Capítulo 3; las nuevas especificaciones de las restricciones permiten aprovechar por completo las capacidades de lenguaje de consulta SQL.

Finalmente, se estudia el concepto de *base de datos activa* en los Apartados 5.8 y 5.9. Una **base de datos activa** tiene un conjunto de **disparadores** que especifica el DBA. Cada disparador describe acciones que se deben llevar a cabo cuando se dan determinadas circunstancias. El SGBD vigila la base de datos, detecta esas situaciones e invoca al disparador. La norma SQL:1999 exige el soporte de los disparadores, y varios productos de SGBD relacionales ya soportan alguna forma de disparador.

<i>idm</i>	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>
22	Domínguez	7	45,0
29	Bravo	1	33,0
31	Lorca	8	55,5
32	Alández	8	25,5
58	Rubio	10	35,0
64	Horacio	7	35,0
71	Zuazo	10	16,0
74	Horacio	9	35,0
85	Arturo	3	25,5
95	Benito	3	63,5

Figura 5.1 El ejemplar *M3* de Marineros

<i>idm</i>	<i>idb</i>	<i>fecha</i>
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figura 5.2 El ejemplar *R2* de Reservas

Acerca de los ejemplos

Se van a presentar varias consultas de ejemplo que emplean la siguiente tabla de definiciones:

Marineros(idm: integer, nombrem: string, categoría: integer, edad: real)
Barcos(idb: integer, nombreb: string, color: string)
Reservas(idm: integer, idb: integer, fecha: date)

A cada consulta se le concede un número único, que continúa el esquema de numeración empleado en el Capítulo 4. La primera consulta nueva de este capítulo tiene el número C15. Las consultas C1 a C14 se introdujeron en el Capítulo 4¹. Las consultas se ilustran mediante los ejemplares *M3* de Marineros, *R2* de Reservas y *B1* de Barcos introducidas en el mismo capítulo, que se reproducen en las Figuras 5.1, 5.2 y 5.3, respectivamente.

Todas las tablas y consultas de ejemplo que aparecen en este capítulo están disponibles en inglés en la página web del libro en

<http://www.cs.wisc.edu/~dbbook>

El material en línea incluye instrucciones sobre la configuración de Oracle, IBM DB2, Microsoft SQL Server y MySQL, y secuencias de comandos para la creación de tablas y consultas de ejemplo.

5.2 FORMA DE LAS CONSULTAS SQL BÁSICAS

Este apartado presenta la sintaxis de las consultas SQL sencillas y explica su significado mediante una *estrategia de evaluación conceptual*. Las estrategias de evaluación conceptual son un medio de evaluar las consultas que se pretende que sean fáciles de comprender antes que eficientes. Los SGBD suelen ejecutar cada consulta de manera diferente y del modo más eficiente.

La forma básica de las consultas SQL es la siguiente:

¹Todas las referencias a cada una de las consultas se pueden hallar en el índice del libro.

<i>idb</i>	<i>nombreb</i>	<i>color</i>
101	Intrépido	azul
102	Intrépido	rojo
103	Campeón	verde
104	Místico	rojo

Figura 5.3 El ejemplar *B1* de Barcos

```
SELECT [ DISTINCT ] lista-de-selección
FROM   lista-de-tablas
WHERE  condición
```

Todas las consultas deben tener una cláusula **SELECT**, que especifica las columnas que se deben conservar en el resultado, y una cláusula **FROM**, que especifica un producto cartesiano de tablas. La cláusula opcional **WHERE** especifica las condiciones de selección para las tablas indicadas en la cláusula **FROM**.

Una consulta así, intuitivamente, corresponde a una expresión del álgebra relacional que implica selecciones, proyecciones y productos cartesianos. La estrecha relación entre SQL y el álgebra relacional es la base de la optimización de las consultas en los SGBD relacionales. En realidad, los planes de ejecución de las consultas SQL se representan mediante una variación de las expresiones del álgebra relacional.

Considérese un ejemplo sencillo.

(C15) *Averiguar el nombre y la edad de todos los marineros.*

```
SELECT DISTINCT M.nombrem, M.edad
FROM   Marineros M
```

La respuesta es un *conjunto* de filas, cada una de ellas es un par $\langle nombrem, edad \rangle$. Si dos o más marineros tienen el mismo nombre y la misma edad, la respuesta sigue teniendo una sola pareja con ese nombre y edad. Esta consulta es equivalente a la aplicación del operador proyección del álgebra relacional.

Si se omite la palabra clave **DISTINCT**, se obtendrá una copia de la fila $\langle n, e \rangle$ por cada marinero de nombre n y edad e ; la respuesta sería un *multiconjunto* de filas. Los **multiconjuntos** se parecen a los conjuntos en que son colecciones desordenadas de elementos, pero en los multiconjuntos puede haber varias copias de cada elemento y el número de copias es significativo —dos multiconjuntos pueden tener los mismos elementos y ser diferentes porque el número de copias de algún elemento sea diferente—. Por ejemplo, $\{a, b, b\}$ y $\{b, a, b\}$ denotan el mismo multiconjunto, y son diferentes del multiconjunto $\{a, a, b\}$.

La respuesta a esta consulta, con o sin la palabra clave **DISTINCT**, para el ejemplar *M3* de *Marineros* puede verse en las Figuras 5.4 y 5.5. La única diferencia es que la tupla de Horacio aparece dos veces si se omite **DISTINCT**; esto se debe a que hay dos marineros que se llaman Horacio y tienen 35 años.

La siguiente consulta es equivalente a una aplicación del operador selección del álgebra relacional.

(C11) *Averiguar todos los marineros de categoría superior a 7.*

<i>nombrem</i>	<i>edad</i>
Domínguez	45,0
Bravo	33,0
Lorca	55,5
Alández	25,5
Rubio	35,0
Horacio	35,0
Zuazo	16,0
Arturo	25,5
Benito	63,5

Figura 5.4 Respuesta a C15

<i>nombrem</i>	<i>edad</i>
Domínguez	45,0
Bravo	33,0
Lorca	55,5
Alández	25,5
Rubio	35,0
Horacio	35,0
Zuazo	16,0
Horacio	35,0
Arturo	25,5
Benito	63,5

Figura 5.5 Respuesta a C15 sin DISTINCT

```
SELECT M.idm, M.nombrém, M.categoría, M.edad
FROM   Marineros AS M
WHERE  M.categoría > 7
```

Esta consulta emplea la palabra clave opcional **AS** para introducir una variable de rango. Por cierto, cuando se desea recuperar todas las columnas como en esta consulta, SQL ofrece una cómoda abreviatura: basta con escribir **SELECT ***. Esta notación resulta útil para las consultas interactivas, pero no es una buena práctica para las consultas que se pretende volver a utilizar y conservar, ya que el esquema del resultado no queda claro desde la propia consulta; hay que hacer referencia al esquema de la tabla *Marineros* subyacente.

Como ilustran estos dos ejemplos, la cláusula **SELECT** se emplea realmente para hacer *proyecciones*, mientras que las *selecciones* en el sentido del álgebra relacional se expresan mediante la cláusula **WHERE**. Este desajuste entre la denominación de los operadores selección y proyección del álgebra relacional y la sintaxis de SQL es un accidente histórico desafortunado.

Ahora se considerará la sintaxis de una consulta básica de SQL con más detalle.

- La **lista-de-tablas** de la cláusula **FROM** es una lista de nombres de tablas. El nombre de cada tabla puede ir seguido de una **variable de rango**; las variables de rango resultan especialmente útiles cuando el mismo nombre de tabla aparece más de una vez en esta lista.
- La **lista-de-selección** es una lista de (expresiones que implican a) nombres de columna de tablas que se mencionan en la lista-de-selección. A los nombres de columna se les puede anteponer una variable de rango.
- La **condición** de la cláusula **WHERE** es una combinación booleana (es decir, una expresión que emplea las conectivas lógicas **AND**, **OR** y **NOT**) de condiciones de la forma *expresión op expresión*, donde *op* es alguno de los operadores de comparación $\{<, \leq, =, >, \geq, >\}$ ².

²Las expresiones con **NOT** siempre se pueden sustituir por expresiones equivalentes sin **NOT** dado el conjunto de operaciones de comparación que se acaba de mencionar.

Una *expresión* es un nombre de *columna*, una *constante* o una expresión (aritmética o cadena de caracteres).

- La palabra clave DISTINCT es opcional. Indica que la tabla calculada como respuesta a la consulta no debe contener *duplicados*, es decir, dos copias de la misma fila. El valor predeterminado es que los duplicados no se eliminan.

Aunque las reglas anteriores describen (de manera informal) la sintaxis de las consultas básicas de SQL, no indican el *significado* de las consultas. La respuesta a cada consulta es en sí misma una relación —que es un *multiconjunto* de filas en SQL— cuyo contenido se puede comprender considerando la siguiente estrategia de evaluación conceptual:

1. Calcular el producto cartesiano de las tablas de la **lista-de-tablas**.
2. Eliminar filas del producto cartesiano que no cumplan las condiciones de **condición**.
3. Eliminar todas las columnas que no aparezcan en la **lista-de-selección**.
4. Si se especifica DISTINCT, eliminar las filas repetidas.

Esta sencilla estrategia de evaluación conceptual explica las filas que deben encontrarse en la respuesta a la consulta. Sin embargo, es probable que sea bastante ineficiente. En capítulos posteriores se considerará la manera en que los SGBD evalúan realmente las consultas; por ahora, nuestro objetivo no es más que explicar el significado de las consultas. La estrategia de evaluación conceptual se ilustrará mediante la consulta siguiente:

(C1) *Averiguar el nombre de los marineros que han reservado el barco 103.*

Se puede expresar en SQL de la manera siguiente.

```
SELECT M.nombrem
  FROM Marineros M, Reservas R
 WHERE M.idm = R.idm AND R.idb=103
```

Calculemos la respuesta a esta consulta para los ejemplares *R3* de Reservas y *M4* de Marineros de las Figuras 5.6 y 5.7, ya que el cálculo para los ejemplares de ejemplo habituales (*R2* y *M3*) resultaría innecesariamente tedioso.

<i>idm</i>	<i>idb</i>	<i>fecha</i>
22	101	10/10/96
58	103	11/12/96

Figura 5.6 Ejemplar *R3* de Reservas

<i>idm</i>	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>
22	domínguez	7	45,0
31	lorca	8	55,5
58	rubio	10	35,0

Figura 5.7 Ejemplar *M4* de Marineros

El primer paso es crear el producto cartesiano $M4 \times R3$, que puede verse en la Figura 5.8.

El segundo paso es aplicar la condición $M.idm = R.idm \text{ AND } R.idb=103$. (Obsérvese que la primera parte de esta condición necesita una operación reunión.) Este paso elimina todas las filas del ejemplar mostrado en la Figura 5.8 menos la última. El tercer paso es eliminar las columnas no deseadas; sólo aparece *nombrem* en la cláusula SELECT. Este paso deja con el resultado mostrado en la Figura 5.9, que es una tabla con una sola columna y, por lo que se ve, una sola fila.

<i>idm</i>	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>	<i>idm</i>	<i>idb</i>	<i>fecha</i>
22	domínguez	7	45,0	22	101	10/10/96
22	domínguez	7	45,0	58	103	11/12/96
31	lorca	8	55,5	22	101	10/10/96
31	lorca	8	55,5	58	103	11/12/96
58	rubio	10	35,0	22	101	10/10/96
58	rubio	10	35,0	58	103	11/12/96

Figura 5.8 $M_4 \times R_3$

<i>nombrem</i>
rubio

Figura 5.9 Respuesta a la consulta C1 para R3 y M4

5.2.1 Ejemplos de consultas básicas de SQL

A continuación se presentarán varias consultas de ejemplo, muchas de las cuales ya se expresaron anteriormente en el álgebra y el cálculo relacionales (Capítulo 4). El primer ejemplo ilustra que el empleo de variables de rango es opcional, a menos que sean necesarias para resolver alguna ambigüedad. La consulta C1, que ya se examinó en el apartado anterior, también puede expresarse de la manera siguiente:

```
SELECT nombrem
  FROM Marineros M, Reservas R
 WHERE M.idm = R.idm AND idb=103
```

Sólo hay que cualificar las apariciones de *idm*, ya que esta columna aparece tanto en la tabla Marineros como en la tabla Reservas. Una manera equivalente de escribir esta consulta es:

```
SELECT nombrem
  FROM Marineros, Reservas
 WHERE Marineros.idm = Reservas.idm AND idb=103
```

Esta consulta muestra que los nombres de las tablas se pueden emplear de manera implícita como variables de fila. Sólo hace falta introducir variables de rango de manera explícita cuando la cláusula **FROM** contiene más de una aparición de una misma relación³. No obstante, se recomienda el empleo explícito de las variables de rango y la condición completa de todas las apariciones de las columnas con una variable de rango para mejorar la legibilidad de las consultas. En todos los ejemplos del libro se seguirá este convenio.

(C16) Averiguar el *idm* de los marineros que han reservado barcos rojos.

³El nombre de la tabla no se puede emplear como variable de rango implícita una vez se ha introducido una variable de rango para la relación.

```
SELECT R.idm
FROM Barcos B, Reservas R
WHERE B.idb = R.idb AND B.color = 'rojo'
```

Esta consulta contiene una reunión de dos tablas, seguida de una selección del color de los barcos. Se puede pensar en B y en R como en filas de las tablas correspondientes que “prueban” que un marinero con idm R.idm reservó un barco rojo B.idb. En los ejemplares de ejemplo *R2* y *M3* (Figuras 5.1 y 5.2), la respuesta consiste en los *idms* 22, 31 y 64. Si se desean los nombres de los marineros en el resultado, también se debe considerar la relación *Marineros*, ya que *Reservas* no contiene esta información, como ilustra el ejemplo siguiente.

(C2) *Averiguar el nombre de los marineros que han reservado barcos rojos.*

```
SELECT M.nombrem
FROM Marineros M, Reservas R, Barcos B
WHERE M.idm = R.idm AND R.idb = B.idb AND B.color = 'rojo'
```

Esta consulta contiene una reunión de tres tablas seguida de una selección del color de los barcos. La reunión con *Marineros* permite averiguar el nombre del marinero que, según la tupla R de *Reservas*, ha reservado el barco rojo descrito por la tupla B.

(C3) *Averiguar el color de los barcos alquilados por Lorca.*

```
SELECT B.color
FROM Marineros M, Reservas R, Barcos B
WHERE M.idm = R.idm AND R.idb = B.idb AND M.nombrem = 'Lorca'
```

Esta consulta es muy parecida a la anterior. Obsérvese que, en general, puede que haya más de un marinero llamado Lorca (ya que *nombrem* no es clave de *Marineros*); esta consulta sigue siendo correcta en el sentido de que devolverá el color de los barcos reservados por *algún* Lorca, si es que hay varios marineros llamados Lorca.

(C4) *Averiguar el nombre de los marineros que han reservado, como mínimo, un barco.*

```
SELECT M.nombrem
FROM Marineros M, Reservas R
WHERE M.idm = R.idm
```

La reunión de *Marineros* y *Reservas* garantiza que, para cada *nombrem* seleccionado, el marinero haya hecho alguna reserva. (Si algún marinero no ha hecho ninguna reserva, el segundo paso de la estrategia de evaluación conceptual eliminará todas las filas del producto cartesiano que impliquen a ese marinero.)

5.2.2 Expresiones y cadenas de caracteres en la orden

SELECT

SQL soporta una versión más general de la **lista-de-selección** que una mera lista de columnas. Cada elemento de una **lista-de-selección** puede ser de la forma *expresión AS nombre_columna*, donde *expresión* es cualquier expresión aritmética o de cadenas de caracteres para los nombres de las columnas (posiblemente con variables de rango antepuestas) y constantes, y *nombre_columna* es un nombre nuevo para esa columna en el resultado de la consulta.

Las expresiones regulares en SQL. Para reflejar la creciente importancia de los datos de texto, SQL:1999 incluye una versión más potente del operador LIKE denominada SIMILAR. Este operador permite emplear una amplia variedad de expresiones regulares como estructuras mientras se busca texto. Las expresiones regulares son parecidas a las soportadas por el sistema operativo Unix para la búsqueda de cadenas de caracteres, aunque la sintaxis es un poco diferente.

También puede contener *agregados* como *sum* y *count*, que se tratarán en el Apartado 5.5. La norma de SQL también incluye expresiones para los valores de fecha y de hora, que no se tratarán aquí. Aunque no forme parte de la norma de SQL, muchas implementaciones soportan también el empleo de funciones predefinidas como *sqrt*, *sen* y *mod*.

(C17) *Calcular el incremento de la categoría de las personas que han navegado en dos barcos diferentes el mismo día.*

```
SELECT M.nombrem, M.categoría+1 AS categoría
FROM   Marineros M, Reservas R1, Reservas R2
WHERE  M.idm = R1.idm AND M.idm = R2.idm
       AND R1.fecha = R2.fecha AND R1.idb <> R2.idb
```

Además, cada elemento de la *condición* puede ser tan general como *expresión1 = expresión2*.

```
SELECT M1.nombrem AS nombre1, M2.nombrem AS nombre2
FROM   Marineros M1, Marineros M2
WHERE  2*M1.categoría = M2.categoría-1
```

Para la comparación de cadenas de caracteres se pueden emplear las operaciones de comparación (=, <, >, etc.) con el orden de las cadenas de caracteres determinado alfabéticamente, como de costumbre. Si hay que ordenar las cadenas de caracteres de manera distinta a la alfabética (por ejemplo, ordenar las cadenas de caracteres que denotan el nombre de los meses según su orden en el calendario enero, febrero, marzo, etc.), SQL soporta el concepto general de **ordenación**, u orden de colocación, para los conjuntos de caracteres. La ordenación permite que el usuario especifique los caracteres que son “menores que” otros y ofrece gran flexibilidad para la manipulación de cadenas de caracteres.

Además, SQL ofrece soporte para la comparación de estructuras mediante el operador LIKE, junto con el uso de los símbolos comodín % (que sustituye a cero o más caracteres arbitrarios) y _ (que sustituye exactamente a un carácter arbitrario). Así, ‘_AB%’ denota una estructura que coincide con todas las cadenas de caracteres que contienen, como mínimo, tres caracteres, en las que el segundo y el tercer carácter son, respectivamente, A y B. Obsérvese que, a diferencia de los demás operadores de comparación, los espacios en blanco pueden resultar significativos para el operador LIKE (dependiendo de la ordenación del conjunto de caracteres subyacente). Por tanto, ‘Jesús’ = ‘Jesús ’ es verdadero, mientras que ‘Jesús’ LIKE ‘Jesús ’ es falso. A continuación se da un ejemplo del empleo de LIKE en una consulta.

(C18) *Averiguar la edad de los marineros cuyo nombre comienza por B, acaba por O y tiene, como mínimo, seis caracteres.*

El álgebra relacional y SQL. Las operaciones de conjuntos de SQL están disponibles en el álgebra relacional. La principal diferencia, por supuesto, es que en SQL se trata de operaciones sobre *multiconjuntos*, ya que las tablas son multiconjuntos de tuplas.

```
SELECT M.edad
FROM   Marineros M
WHERE  M.nombrem LIKE 'B_%__O'
```

El único marinero así es Benito, y su edad es 63,5.

5.3 UNION, INTERSECT Y EXCEPT

SQL ofrece tres estructuras para la manipulación de conjuntos que amplían la forma básica de las consultas presentadas anteriormente. Dado que la respuesta a cada consulta es un multiconjunto de filas, resulta natural considerar el empleo de operaciones como la unión, la intersección y la diferencia. SQL soporta estas operaciones con el nombre de **UNION**, **INTERSECT** y **EXCEPT**⁴. SQL ofrece también otras operaciones con conjuntos: **IN** (para comprobar si un elemento pertenece a un conjunto dado), **op ANY**, **op ALL** (para comparar un valor con los elementos de un conjunto dado, mediante el operador comparación **op**) y **EXISTS** (para comprobar si un conjunto está vacío). **IN** y **EXISTS** pueden llevar antepuesto **NOT**, con la modificación obvia de su significado. En este apartado se tratarán **UNION**, **INTERSECT** y **EXCEPT**, y las demás operaciones se tratarán en el Apartado 5.4.

Considérese la consulta siguiente:

(C5) *Averiguar el nombre de los marineros que han reservado barcos rojos o verdes.*

```
SELECT M.nombrem
FROM   Marineros M, Reservas R, Barcos B
WHERE  M.idm = R.idm AND R.idb = B.idb
       AND (B.color = 'rojo' OR B.color = 'verde')
```

Esta consulta se expresa fácilmente mediante la conectiva **OR** de la cláusula **WHERE**. Sin embargo, la consulta siguiente, que es idéntica salvo por el empleo de “y” en lugar de “o” en su versión en castellano, resulta ser mucho más difícil:

(C6) *Averiguar el nombre de los marineros que han reservado tanto barcos rojos como barcos verdes.*

Si simplemente hubiera que sustituir el empleo de **OR** en la consulta anterior por el de **AND**, en analogía con la expresión en castellano de las dos consultas, se recuperaría el nombre de los marineros que han reservado barcos que son a la vez rojos y verdes. La restricción de integridad de que *idb* es clave de Barcos indica que el mismo barco no puede tener dos colores y, por tanto, la variante de la consulta anterior con **AND** en lugar de **OR** devolverá siempre un conjunto de respuestas vacío. La expresión correcta de la consulta C6 mediante **AND** es la siguiente:

⁴Obsérvese que, aunque la norma de SQL incluye estas operaciones, muchos sistemas actualmente sólo soportan **UNION**. Además, muchos sistemas reconocen la palabra clave **MINUS** en lugar de **EXCEPT**.

```

SELECT M.nombrem
FROM   Marineros M, Reservas R1, Barcos B1, Reservas R2, Barcos B2
WHERE  M.idm = R1.idm AND R1.idb = B1.idb
       AND M.idm = R2.idm AND R2.idb = B2.idb
       AND B1.color='rojo' AND B2.color = 'verde'

```

Se puede pensar en R1 y en B1 como en filas que prueban que el marinero M.idm ha reservado barcos rojos. R2 y B2, de manera parecida, prueban que el mismo marinero ha reservado barcos verdes. M.nombrem no se incluye en el resultado a menos que se encuentren cinco filas como M, R1, B1, R2 y B2.

La consulta anterior es difícil de comprender (y también ineficiente de ejecutar, como puede verse). En especial, el parecido con la consulta anterior **OR** (la consulta C5) se pierde completamente. Una solución más adecuada para estas dos consultas es emplear **UNION** e **INTERSECT**.

La consulta **OR** (la consulta C5) se puede reescribir de la manera siguiente:

```

SELECT M.nombrem
FROM   Marineros M, Reservas R, Barcos B
WHERE  M.idm = R.idm AND R.idb = B.idb AND B.color = 'rojo'
UNION
SELECT M2.nombrem
FROM   Marineros M2, Barcos B2, Reservas R2
WHERE  M2.idm = R2.idm AND R2.idb = B2.idb AND B2.color = 'verde'

```

Esta consulta indica que se desea la unión del conjunto de marineros que han reservado barcos rojos con el conjunto de marineros que han reservado barcos verdes. En completa simetría, la consulta **AND** (la consulta C6) se puede reescribir de la manera siguiente:

```

SELECT M.nombrem
FROM   Marineros M, Reservas R, Barcos B
WHERE  M.idm = R.idm AND R.idb = B.idb AND B.color = 'rojo'
INTERSECT
SELECT M2.nombrem
FROM   Marineros M2, Barcos B2, Reservas R2
WHERE  M2.idm = R2.idm AND R2.idb = B2.idb AND B2.color = 'verde'

```

Esta consulta contiene realmente un fallo sutil —si hay dos marineros como Horacio en los ejemplares de ejemplo *B1*, *R2* y *M3*, uno de los cuales ha reservado un barco rojo, mientras que el otro ha reservado un barco verde, se devolverá el nombre de Horacio aunque ninguna persona llamada Horacio haya reservado tanto barcos rojos como verdes—. Por tanto, la consulta calcula realmente nombres de marineros tales que algún marinero con ese nombre haya reservado un barco rojo y algún marinero con el mismo nombre (acaso un marinero diferente) haya reservado un barco verde.

Como se observó en el Capítulo 4, el problema surge porque se emplea *nombrem* para identificar a los marineros, y *nombrem* no es clave de *Marineros*. Si se selecciona *idm* en lugar de *nombrem* en la consulta anterior, se calculará el conjunto de *idms* de marineros que han reservado tanto barcos rojos como barcos verdes. (Para calcular el nombre de esos marineros hace falta una consulta anidada; se volverá a este ejemplo en el Apartado 5.4.4.)

La siguiente consulta ilustra la operación diferencia de conjuntos en SQL.

(C19) Averiguar el idm de todos los marineros que han reservado barcos rojos pero no verdes.

```
SELECT M.idm
FROM Marineros M, Reservas R, Barcos B
WHERE M.idm = R.idm AND R.idb = B.idb AND B.color = 'rojo'
EXCEPT
SELECT M2.idm
FROM Marineros M2, Reservas R2, Barcos B2
WHERE M2.idm = R2.idm AND R2.idb = B2.idb AND B2.color = 'verde'
```

Los marineros 22, 64 y 31 han reservado barcos rojos. Los marineros 22, 74 y 31 han reservado barcos verdes. Por tanto, la respuesta contiene sólo el *idm* 64.

En realidad, dado que la relación Reservas contiene información sobre los *idms*, no hay necesidad alguna de consultar la relación Marineros, y se puede emplear la consulta siguiente, que es más sencilla:

```
SELECT R.idm
FROM Barcos B, Reservas R
WHERE R.idb = B.idb AND B.color = 'rojo'
EXCEPT
SELECT R2.idm
FROM Barcos B2, Reservas R2
WHERE R2.idb = B2.idb AND B2.color = 'verde'
```

Obsérvese que esta consulta confía en la integridad referencial; es decir, no hay reservas de marineros que no existan. Téngase en cuenta que UNION, INTERSECT y EXCEPT se pueden emplear en *cualesquiera* dos tablas que sean compatibles para la unión, es decir, tienen el mismo número de columnas y esas columnas, tomadas en orden, tienen el mismo tipo. Por ejemplo, se puede escribir la consulta siguiente:

(C20) Averiguar el idm de los marineros que tengan una categoría de 10 o hayan reservado el barco 104.

```
SELECT M.idm
FROM Marineros M
WHERE M.categoría = 10
UNION
SELECT R.idm
FROM Reservas R
WHERE R.idb = 104
```

La primera parte de la unión devuelve los *idms* 58 y 71. La segunda parte devuelve 22 y 31. La respuesta es, por tanto, el conjunto de *idms* 22, 31, 58 y 71. Un último punto a destacar sobre UNION, INTERSECT y EXCEPT es el siguiente. A diferencia del criterio predeterminado de que en la forma básica de las consultas no se eliminan los duplicados a menos que se especifique DISTINCT, el criterio predeterminado para las consultas con UNION es que los duplicados *sí* se eliminan. Para conservar los duplicados se debe emplear UNION ALL; en ese

El álgebra relacional y SQL. El anidamiento de consultas es una característica que no está disponible en el álgebra relacional, pero las consultas anidadas se pueden traducir al álgebra. El anidamiento en SQL está más inspirado por el cálculo relacional que por el álgebra. Junto con algunas otras características de SQL, como los operadores para (multi)conjuntos y la agregación, el anidamiento es una estructura muy expresiva.

caso, el número de copias de cada fila del resultado es siempre $m + n$, donde m y n son el número de veces que la fila aparece en las dos partes de la unión. De manera parecida, **INTERSECT ALL** conserva los duplicados —el número de copias de cada fila del resultado es $\min(m, n)$ — y **EXCEPT ALL** también conserva los duplicados —el número de copias de cada fila del resultado es $m - n$, donde m corresponde a la primera relación—.

5.4 CONSULTAS ANIDADAS

Una de las características más potentes de SQL son las consultas anidadas. Una **consulta anidada** es una consulta que tiene otra consulta incorporada en su interior; la consulta incorporada se denomina **subconsulta**. Por supuesto, la propia consulta incorporada puede ser una consulta anidada; por tanto, se pueden encontrar consultas que tengan estructuras anidadas muy profundas. Al escribir una consulta a veces hay que expresar una condición que hace referencia a alguna tabla que, a su vez, se debe calcular. Las consultas empleadas para calcular esas tablas subsidiarias son subconsultas y aparecen como parte de las consultas principales. Las subconsultas suelen aparecer en el interior de las cláusulas **WHERE** de las consultas. Las subconsultas pueden aparecer a veces en las cláusulas **FROM** o **HAVING** (que se presentarán en el Apartado 5.5). Este apartado sólo estudia las subconsultas que aparecen en las cláusulas **WHERE**. El tratamiento de las subconsultas que aparecen en otras partes de las consultas es muy parecido. Algunos ejemplos de subconsulta que aparecen en la cláusula **FROM** se estudian posteriormente en el Apartado 5.5.1.

5.4.1 Introducción a las consultas anidadas

A manera de ejemplo se reescribirá la consulta siguiente, examinada previamente, empleando una subconsulta anidada:

(C1) *Averiguar el nombre de los marineros que han reservado el barco 103.*

```
SELECT M.nombrem
  FROM Marineros M
 WHERE M.idm IN ( SELECT R.idm
                      FROM Reservas R
                     WHERE R.idb = 103 )
```

La subconsulta anidada calcula el (multi)conjunto de *idms* de los marineros que han reservado el barco 103 (el conjunto contiene 22, 31 y 74 para los ejemplares *R2* y *M3*) y la consulta de nivel superior recupera el nombre de los marineros cuyo *idm* se halla en ese conjunto. El operador **IN** permite comprobar si un valor pertenece a un conjunto de elementos

dado; para generar el conjunto que se desea comprobar se utiliza una consulta SQL. Obsérvese que resulta muy sencillo modificar esta consulta para averiguar todos los marineros que *no* han reservado el barco 103 —basta con sustituir `IN` por `NOT IN`—.

La mejor manera de comprender una consulta anidada es pensar en ella en términos de una estrategia de evaluación conceptual. En este ejemplo, la estrategia consiste en examinar las filas de *Marineros* y para cada una de ellas evaluar la subconsulta para *Reservas*. En general, la estrategia de evaluación conceptual que se ha presentado para definir la semántica de una consulta se puede ampliar para que abarque las consultas anidadas de la manera siguiente: se crea el producto cartesiano de las tablas de la cláusula `FROM` de la consulta de nivel anterior como se hizo anteriormente. Por cada fila del producto cartesiano, mientras se comprueba la condición en la cláusula `WHERE`, se (vuelve a) calcular la subconsulta⁵. Por supuesto, la propia subconsulta podría contener otra subconsulta anidada, en cuyo caso se aplicaría nuevamente la misma idea, lo que llevaría a una estrategia de evaluación con varios niveles de bucles anidados.

Como ejemplo de consulta con varios anidamientos, se reescribirá la consulta siguiente.

(C2) Averiguar el nombre de los marineros que han reservado barcos rojos.

```
SELECT M.nombrem
  FROM Marineros M
 WHERE M.idm IN ( SELECT R.idm
                      FROM Reservas R
                     WHERE R.idb IN ( SELECT B.idb
                                      FROM Barcos B
                                     WHERE B.color = 'rojo' ) )
```

La subconsulta más interna busca el conjunto de *idbs* de los barcos rojos (102 y 104 para el ejemplar *B1*). La subconsulta que se halla un nivel por encima de ella busca el conjunto de *idms* de marineros que han reservado alguno de esos barcos. Para los ejemplares *B1*, *R2* y *M3* ese conjunto de *idms* contiene 22, 31 y 64. La consulta de nivel superior busca el nombre de los marineros cuyo *idm* pertenece a ese conjunto de *idms*; se obtiene Domínguez, Lorca y Horacio.

Para averiguar el nombre de los marineros que no han reservado ningún barco rojo, se sustituye la aparición más externa de `IN` por `NOT IN`, como se ilustra en la consulta siguiente.

(C21) Averiguar el nombre de los marineros que no han reservado ningún barco rojo.

```
SELECT M.nombrem
  FROM Marineros M
 WHERE M.idm NOT IN ( SELECT R.idm
                      FROM Reservas R
                     WHERE R.idb IN ( SELECT B.idb
                                      FROM Barcos B
                                     WHERE B.color = 'rojo' ) )
```

⁵Dado que la subconsulta interior del ejemplo no depende de la fila “actual” de la consulta externa de ninguna manera, uno se podría preguntar el motivo de tener que calcular de nuevo la subconsulta para cada fila externa. Para entender el porqué véase el Apartado 5.4.2.

Esta consulta calcula el nombre de los marineros cuyo *idm* no pertenece al conjunto 22, 31 y 64.

A diferencia de la consulta C21, se puede modificar la consulta anterior (la versión anidada de C2) sustituyendo la aparición interior (en vez de la aparición exterior) de **IN** por **NOT IN**. Esta consulta modificada calculará el nombre de los marineros que han reservado barcos que no son rojos, es decir, si han hecho alguna reserva, no es de ningún barco rojo. Considérese la manera de hacerlo. En la consulta interior se comprueba que *R.idb* no es ni 102 ni 104 (los *idbs* de los barcos rojos). La consulta exterior busca entonces los *idms* de las tuplas de Reservas en las que el *idb* no es ni 102 ni 104. Para los ejemplares *B1*, *R2* y *M3*, la consulta exterior calcula el conjunto de *idms* 22, 31, 64 y 74. Finalmente, se averigua el nombre de los marineros cuyo *idm* pertenece a ese conjunto.

También se puede modificar la consulta anidada C2 sustituyendo las dos apariciones de **IN** por **NOT IN**. Esta variante busca el nombre de los marineros que no han reservado barcos que no sean rojos, es decir, que sólo han reservado barcos rojos (si es que han reservado algún barco). Actuando como en el párrafo anterior, para los ejemplares *B1*, *R2* y *M3*, la consulta exterior calcula el conjunto de *idms* (de Marineros) que no son 22, 31, 64 ni 74. Se trata del conjunto 29, 32, 58, 71, 85 y 95. Luego se busca el nombre de los marineros cuyo *idm* pertenece a ese conjunto.

5.4.2 Consultas anidadas correlacionadas

En las consultas anidadas vistas hasta el momento la subconsulta interior ha sido completamente independiente de la consulta exterior. En general, la subconsulta interior puede depender de la fila que se está examinando en cada momento en la consulta exterior (en términos de la estrategia de evaluación conceptual). Reescribamos una vez más la siguiente consulta.

(C1) Averiguar el nombre de los marineros que han reservado el barco número 103.

```
SELECT M.nombrem
FROM   Marineros M
WHERE  EXISTS ( SELECT *
                  FROM   Reservas R
                  WHERE  R.idb = 103
                          AND R.idm = M.idm )
```

El operador **EXISTS** es otro operador para la comparación de conjuntos, como **IN**. Permite comprobar si un conjunto no está vacío, es decir, se trata de una comparación implícita de igualdad con el conjunto vacío. Por tanto, para cada fila *M* de Marineros, se comprueba si el conjunto de filas de Reservas *R* tal que *R.idb = 103 AND M.idm = R.idm* no está vacío. Si no lo está, el marinero *M* ha reservado el barco 103 y se devuelve su nombre. La subconsulta depende claramente de la fila actual *M* y se debe volver a evaluar para cada fila de Marineros. La aparición de *M* en la subconsulta (en forma del literal *M.idm*) se denomina *correlación*, y estas consultas se denominan *consultas correlacionadas*.

Esta consulta también ilustra el empleo del símbolo especial * en situaciones en las que todo lo que se desea hacer es comprobar si existe una fila que cumpla la condición y no se desea realmente recuperar ninguna columna de esa fila. Éste es uno de los dos empleos de *

en la cláusula **SELECT** que se considera buen estilo de programación; el otro es un argumento de la operación de agregación **COUNT**, que se describirá en breve.

Como ejemplo adicional, mediante el empleo de **NOT EXISTS** en lugar de **EXISTS**, se puede calcular el nombre de los marineros que no han reservado barcos rojos. Estrechamente relacionado con **EXISTS** está el predicado **UNIQUE**. Cuando se aplica **UNIQUE** a una subconsulta, la condición resultante devuelve **verdadero** si ninguna fila aparece dos veces en la respuesta de la subconsulta, es decir, si no hay ningún duplicado; en especial, devuelve **verdadero** si la respuesta está vacía. (También hay una versión **NOT UNIQUE**.)

5.4.3 Operadores para la comparación de conjuntos

Ya se han visto los operadores para comparación de conjuntos **EXISTS**, **IN** y **UNIQUE**, junto con sus versiones negadas. SQL también soporta **op ANY** y **op ALL**, donde **op** es alguno de los operadores de comparación $\{<, \leq, =, >, \geq, >\}$. (También está disponible **SOME**, pero no es más que un sinónimo de **ANY**.)

(C22) Averiguar los marineros cuya categoría es superior a la de algún marinero llamado Horacio.

```
SELECT M.idm
FROM Marineros M
WHERE M.categoría > ANY ( SELECT M2.categoría
                            FROM Marineros M2
                            WHERE M2.nombrem = 'Horacio' )
```

Si hay varios marineros llamados Horacio, esta consulta halla todos los marineros cuya categoría es superior que la de *algún* marinero llamado Horacio. Para el ejemplar *M3*, esto supone los *idms* 31, 32, 58, 71 y 74. ¿Qué ocurriría si *no* hubiera ningún marinero llamado Horacio? En ese caso la comparación *M.categoría > ANY ...* devuelve por definición **falso**, y la consulta devuelve un conjunto de respuestas vacío. Para comprender las comparaciones que emplean **ANY** resulta útil considerar que la comparación se ejecuta de manera reiterada. En este ejemplo, *M.categoría* se compara con éxito con cada valor de la categoría que constituye una respuesta de la consulta anidada. De manera intuitiva, la subconsulta debe devolver una fila que haga que la comparación sea **verdadera**, con objeto de que *M.categoría > ANY ...* devuelva **verdadero**.

(C23) Averiguar los marineros cuya categoría es superior a la de todos los marineros llamados Horacio.

Todas esas consultas se pueden conseguir con una sencilla modificación de la consulta C22: basta con sustituir **ANY** por **ALL** en la cláusula **WHERE** de la consulta exterior. Para el ejemplar *M3* se obtienen los *idms* 58 y 71. Si no hubiera ningún marinero llamado Horacio, la comparación *M.categoría > ALL ...* devuelve por definición **verdadero**, por lo que la consulta devolvería el nombre de todos los marineros. Una vez más resulta útil pensar que la comparación se lleva a cabo de manera reiterada. De manera intuitiva, la comparación debe ser verdadera para todas las filas devueltas para que *M.categoría > ALL ...* devuelva **verdadero**.

Como ilustración adicional de **ALL** considérese la consulta siguiente.

(C24) Averiguar los marineros que tienen la máxima categoría.

```

SELECT M.idm
FROM Marineros M
WHERE M.categoría >= ALL ( SELECT M2.categoría
                            FROM Marineros M2 )

```

La subconsulta calcula el conjunto de todos los valores de categoría de Marineros. La condición exterior WHERE sólo se satisface cuando *M.categoría* es mayor o igual que cada uno de esos valores de categoría, es decir, cuando es el valor de categoría más elevado. En el ejemplar *M3* la condición sólo se satisface para la *categoría* 10, y la respuesta incluye los *idms* de los marineros con esa categoría, es decir, 58 y 71.

Obsérvese que IN y NOT IN son equivalentes a = ANY y <> ALL, respectivamente.

5.4.4 Más ejemplos de consultas anidadas

Revisemos una consulta que se tomó en consideración anteriormente empleando el operador INTERSECT.

(C6) *Averiguar el nombre de los marineros que han reservado tanto barcos rojos como barcos verdes.*

```

SELECT M.nombrem
FROM Marineros M, Reservas R, Barcos B
WHERE M.idm = R.idm AND R.idb = B.idb AND B.color = 'rojo'
      AND M.idm IN ( SELECT M2.idm
                      FROM Marineros M2, Barcos B2, Reservas R2
                      WHERE M2.idm = R2.idm AND R2.idb = B2.idb
                            AND B2.color = 'verde' )

```

Esta consulta se puede entender de la manera siguiente: “Averiguar todos los marineros que han reservado barcos rojos y, además, tienen *idms* que están incluidos en el conjunto de *idms* de marineros que han reservado barcos verdes.” Esta formulación de la consulta ilustra la manera en que las consultas que implican a INTERSECT se pueden reescribir empleando IN, lo que resulta útil saber si el sistema no soporta INTERSECT. Las consultas que emplean EXCEPT se pueden reescribir de manera parecida empleando NOT IN. Para averiguar los *idms* de los marineros que han reservado barcos rojos pero no barcos verdes basta con sustituir la palabra clave IN de la consulta anterior por NOT IN.

Como puede verse, escribir la consulta (C6) empleando INTERSECT resulta más complicado porque hay que utilizar los *idms* para identificar a los marineros (durante la intersección) y hay que devolver el nombre de los marineros:

```

SELECT M.nombrem
FROM Marineros M
WHERE M.idm IN (( SELECT R.idm
                     FROM Barcos B, Reservas R
                     WHERE R.idb = B.idb AND B.color = 'rojo' )
                  INTERSECT
                  (SELECT R2.idm
                     FROM Barcos B2, Reservas R2
                     WHERE R2.idb = B2.idb AND B2.color = 'verde' ))

```

Funciones de agregación de SQL:1999. El conjunto de funciones de agregación se amplía enormemente en la nueva norma, que incluye varias funciones estadísticas como la desviación normal, la covarianza y los percentiles. No obstante, las nuevas funciones de agregación se encuentran en el paquete SQL/OLAP y puede que no todos los fabricantes las soporten.

```
WHERE R2.idb = B2.idb AND B2.color = 'verde' ))
```

El siguiente ejemplo ilustra la manera en que se puede expresar en SQL la operación *división* del álgebra relacional.

(C9) *Averiguar el nombre de los marineros que han reservado todos los barcos.*

```
SELECT M.nombrem
FROM Marineros M
WHERE NOT EXISTS (( SELECT B.idb
                      FROM Barcos B )
EXCEPT
(SELECT R.idb
      FROM Reservas R
     WHERE R.idm = M.idm ))
```

Obsérvese que esta consulta está correlacionada —para cada marinero M se comprueba si el conjunto de barcos reservado por M incluye todos los barcos—. Una manera alternativa de llevar a cabo esta consulta sin emplear EXCEPT es la siguiente:

```
SELECT M.nombrem
FROM Marineros M
WHERE NOT EXISTS ( SELECT B.idb
                      FROM Barcos B
                     WHERE NOT EXISTS ( SELECT R.idb
                                         FROM Reservas R
                                         WHERE R.idb = B.idb
                                         AND R.idm = S.idm ))
```

De manera intuitiva, para cada marinero se comprueba que no hay ningún barco que no haya sido reservado por ese marinero.

5.5 OPERADORES DE AGREGACIÓN

Además de recuperar simplemente datos, a menudo se desea llevar a cabo algún cálculo o resumen. Como ya se ha indicado en este capítulo, SQL permite el empleo de expresiones aritméticas. Consideremos ahora una potente clase de estructuras para el cálculo de *valores de agregación* como MIN y SUM. Estas características representan una ampliación significativa del álgebra relacional. SQL soporta cinco operaciones de agregación, que se pueden aplicar a cualquier columna, como A, de una relación dada:

1. COUNT ([DISTINCT] A): el número de valores (únicos) de la columna A.
2. SUM ([DISTINCT] A): la suma de todos los valores (únicos) de la columna A.
3. AVG ([DISTINCT] A): el promedio de todos los valores (únicos) de la columna A.
4. MAX (A): el valor máximo de la columna A.
5. MIN (A): el valor mínimo de la columna A.

Obsérvese que no tiene sentido especificar DISTINCT en conjunción con MIN o MAX (aunque SQL no lo impida).

(C25) Averiguar el promedio de edad de todos los marineros.

```
SELECT AVG (M.edad)
FROM   Marineros M
```

Para el exemplar *M3* el promedio de edad es 37,4. Por supuesto, se puede utilizar la cláusula WHERE para restringir los marineros que se emplean en el cálculo de la edad promedio.

(C26) Averiguar el promedio de edad de los marineros con categoría 10.

```
SELECT AVG (M.edad)
FROM   Marineros M
WHERE  M.categoría = 10
```

Hay dos marineros así, y su promedio de edad es 25,5. Se pueden emplear MIN (o MAX) en lugar de AVG en las consultas anteriores para averiguar la edad del marinero más joven (o más viejo). Sin embargo, hallar tanto el nombre como la edad del marinero más anciano resulta más complicado, como ilustra la consulta siguiente.

(C27) Averiguar el nombre y la edad del marinero más anciano.

Considérese el siguiente intento de responder a esta consulta:

```
SELECT M.nombrem, MAX (M.edad)
FROM   Marineros M
```

La pretensión es que esta consulta no sólo devuelva la edad máxima, sino también el nombre de los marineros que tienen esa edad. Sin embargo, esta consulta es ilegal en SQL — si la cláusula SELECT emplea alguna operación de agregación, *only* debe emplear operaciones de agregación, a menos que la consulta contenga alguna cláusula GROUP BY—. (La intuición subyacente a esta restricción debería quedar clara cuando se estudie la cláusula GROUP BY en el Apartado 5.5.1.) Por tanto, no se puede emplear MAX (M.edad) ni M.nombrem en la cláusula SELECT. Hay que emplear consultas anidadas para calcular la respuesta deseada a C27:

```
SELECT M.nombrem, M.edad
FROM   Marineros M
WHERE  M.edad = ( SELECT MAX (M2.edad)
                  FROM Marineros M2 )
```

Obsérvese que se ha empleado el resultado de una operación de agregación en la subconsulta como argumento de una operación de comparación. Hablando estrictamente, estamos comparando un valor de edad con el resultado de la subconsulta, que es una relación. Sin embargo, debido al empleo de la operación de agregación, se garantiza que la subconsulta devuelva una sola tupla con un solo campo, y SQL convierte esa relación en un valor de campo con vistas a la comparación. La siguiente consulta equivalente a C27 es legal en la norma de SQL pero, desafortunadamente, muchos sistemas no la soportan:

```
SELECT M.nombrem, M.edad
  FROM Marineros M
 WHERE ( SELECT MAX (M2.edad)
      FROM Marineros M2 ) = M.edad
```

Se puede contar el número de marineros con COUNT. Este ejemplo ilustra el empleo de * como argumento de COUNT, que resulta útil cuando se desea contar todas las filas.

(C28) *Contar el número de marineros.*

```
SELECT COUNT (*)
  FROM Marineros M
```

Se puede considerar * una abreviatura de todas las columnas (del producto cartesiano de la **lista-de-tablas** de la cláusula FROM). Compárese esta consulta con la siguiente, que calcula el número de nombres de marinero diferentes. (Recuérdese que *nombrem* no es clave.)

(C29) *Contar el número de nombres de marinero diferentes.*

```
SELECT COUNT ( DISTINCT M.nombrem )
  FROM Marineros M
```

Para el ejemplar *M3* la respuesta a C28 es 10, mientras que la respuesta a C29 es 9 (ya que dos marineros tienen el mismo nombre, Horacio). Si se omite DISTINCT, la respuesta a C29 es 10, ya que el nombre Horacio se cuenta dos veces. Si COUNT no incluye DISTINCT, entonces COUNT(*) da la misma respuesta que COUNT(*x*), donde *x* es cualquier conjunto de atributos. En el ejemplo anterior y sin DISTINCT, C29 es equivalente a C28. No obstante, el empleo de COUNT (*) supone un mejor estilo de consulta, ya que inmediatamente queda claro que todos los registros contribuyen a la cuenta total.

Las operaciones de agregación ofrecen una alternativa a las estructuras ANY y ALL. Por ejemplo, considérese la consulta siguiente:

(C30) *Averiguar el nombre de los marineros de más edad que el marinero más viejo de categoría 10.*

```
SELECT M.nombrem
  FROM Marineros M
 WHERE M.edad > ( SELECT MAX ( M2.edad )
      FROM Marineros M2
     WHERE M2.categoría = 10 )
```

Para el ejemplar *M3* el marinero de más edad de categoría 10 es el marinero 58, cuya edad es 35. Los nombres de los marineros de más edad son Benito, Domínguez, Horacio y Lorca. Empleando ALL esta consulta se podría escribir también de la manera siguiente:

El álgebra relacional y SQL. La agregación es una operación fundamental que no se puede expresar en el álgebra relacional. De manera parecida, la estructura de agrupamiento de SQL no se puede expresar en el álgebra.

```
SELECT M.nombrem
  FROM Marineros M
 WHERE M.edad > ALL ( SELECT M2.edad
                           FROM Marineros M2
                          WHERE M2.categoría = 10 )
```

Sin embargo, la consulta ALL es más susceptible de sufrir errores —fácil (e incorrectamente) se podría emplear ANY en lugar de ALL y recuperar marineros de más edad que *algún* marinero de categoría 10—. El empleo de ANY se corresponde intuitivamente con el de MIN, en lugar de MAX, en la consulta anterior.

5.5.1 Las cláusulas GROUP BY y HAVING

Hasta ahora se han aplicado las operaciones de agregación a todas las filas (que tenían las condiciones adecuadas) de la relación. A menudo se desea aplicar operaciones de agregación a cada uno de los **grupos** de filas de una relación, donde el número de grupos depende del ejemplar de esa relación (es decir, no se conoce con antelación). Por ejemplo, considérese la consulta siguiente.

(C31) *Averiguar la edad del marinero más joven de cada categoría.*

Si se sabe que las categorías son enteros que van del 1 al 10, se pueden escribir diez consultas de esta forma:

```
SELECT MIN (M.edad)
  FROM Marineros M
 WHERE M.categoría = i
```

donde $i = 1, 2, \dots, 10$. Escribir diez consultas así resulta tedioso. Lo que es más importante, puede que no se conozca con antelación las categorías que hay.

Para escribir estas consultas hace falta una importante extensión de la forma básica de consulta SQL: la cláusula GROUP BY. De hecho, la ampliación también incluye la cláusula opcional HAVING, que se puede emplear para especificar condiciones para los grupos (por ejemplo, puede que sólo interesen categorías > 6). La forma general de una consulta SQL con estas extensiones es:

```
SELECT [ DISTINCT ] lista-de-selección
  FROM lista-de-tablas
 WHERE condición
 GROUP BY lista-para-formar-grupos
 HAVING condición-sobre-grupos
```

Mediante la cláusula GROUP BY se puede escribir C31 de la manera siguiente:

```
SELECT M.categoría, MIN (M.edad)
FROM Marineros M
GROUP BY M.categoría
```

Consideremos algunos aspectos importantes de las nuevas cláusulas:

- La **lista-de-selección** de la cláusula SELECT consta de (1) una lista de nombres de columna y (2) una lista de términos que tienen la forma **opagr** (*nombre-columna*) **AS** *nombre-nuevo*. Ya se vió el empleo de AS para cambiar el nombre a las columnas del resultado. Las columnas que forman el resultado de los operadores de agregación no tienen todavía nombre de columna y, por tanto, resulta especialmente útil darles nombre con AS.

Todas las columnas que aparecen en (1) deben aparecer también en **lista-para-formar-grupos**. El motivo es que cada fila del resultado de la consulta se corresponde con un *grupo*, que es un conjunto de filas que concuerdan con los valores de las columnas de **lista-para-formar-grupos**. En general, si una columna aparece en la lista (1), pero no en **lista-para-formar-grupos**, puede haber varias filas de un mismo grupo que tengan valores diferentes en esa columna, y no resulta evidente el valor que se le debe asignar en la fila de la respuesta.

A veces se puede utilizar información de la clave primaria para comprobar que una columna dada tiene un valor único en todas las filas de cada grupo. Por ejemplo, si **lista-para-formar-grupos** contiene la clave primaria de una tabla de la **lista-de-tablas**, todas las columnas de esta tabla tendrán un valor único dentro de cada grupo. En SQL:1999 también se permite que esas columnas aparezcan en la parte (1) de la **lista-de-selección**.

- Las expresiones que aparecen en **grupo-condición** de la cláusula HAVING deben tener un *solo* valor por grupo. La intuición es que la cláusula HAVING determina si hay que generar una fila de respuesta para un grupo dado. Para satisfacer este requisito en SQL-92, las columnas que aparecen en **grupo-condición** deben aparecer como argumentos del operador de agregación, o deben aparecer también en **lista-para-formar-grupos**. En SQL:1999 se han introducido dos nuevas funciones para conjuntos que permiten comprobar si *todas* o *alguna* de las filas de un grupo dado satisfacen una condición determinada; esto permite emplear condiciones parecidas a las de la cláusula WHERE.
- Si se omite GROUP BY, se considera a toda la tabla como un solo grupo.

La semántica de estas consultas explica mediante un ejemplo.

(C32) Averiguar la edad del marinero más joven que tiene derecho a voto (es decir, tiene, como mínimo, 18 años) para cada categoría que tenga, como mínimo, dos marineros con derecho a voto.

```
SELECT M.categoría, MIN (M.edad) AS edadmín
FROM Marineros M
WHERE M.edad >= 18
GROUP BY M.categoría
HAVING COUNT (*) > 1
```

Esta consulta se evaluará con el ejemplar *M3* de Marineros que se reproduce por comodidad en la Figura 5.10. Para la ampliación de la estrategia de evaluación conceptual del Apartado 5.2 se procede de la manera siguiente. El primer paso es construir el producto cartesiano de las tablas de la **lista-de-tablas**. Como la única relación de la **lista-de-tablas** de la consulta C32 es Marineros, el resultado es el ejemplar mostrado en la Figura 5.10.

<i>idm</i>	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>
22	Domínguez	7	45,0
29	Bravo	1	33,0
31	Lorca	8	55,5
32	Alández	8	25,5
58	Rubio	10	35,0
64	Horacio	7	35,0
71	Zuazo	10	16,0
74	Horacio	9	35,0
85	Arturo	3	25,5
95	Benito	3	63,5
96	Francisco	3	25,5

Figura 5.10 El ejemplar *M3* de Marineros

El segundo paso es aplicar la condición de la cláusula WHERE, $M.edad \geq 18$. Este paso elimina la fila $\langle 71, zuazo, 10, 16 \rangle$. El tercer paso es eliminar las columnas no deseadas. Sólo son necesarias las columnas mencionadas en la cláusula SELECT, la cláusula GROUP BY o la cláusula HAVING, lo que significa que se pueden eliminar *idm* y *nombrem* del ejemplo. El resultado puede verse en la Figura 5.11. Obsérvese que hay dos filas idénticas con *categoría* 3 y *edad* 25,5 —SQL no elimina los duplicados excepto cuando se le obliga a hacerlo mediante el empleo de la palabra clave DISTINCT—. El número de copias de cada fila en la tabla intermedia de la Figura 5.11 viene determinado por el número de filas de la tabla original que tenían esos valores en las columnas proyectadas.

El cuarto paso es ordenar la tabla de acuerdo con la cláusula GROUP BY para identificar los grupos. El resultado de este paso puede verse en la Figura 5.12.

El quinto paso es aplicar la condición de grupos de la cláusula HAVING, es decir, la condición COUNT (*) > 1. Este paso elimina los grupos con *categoría* igual a 1, 9 y 10. Obsérvese que el orden en que se toman en consideración las cláusulas WHERE y GROUP BY es significativo: si no se tomara en consideración en primer lugar la cláusula WHERE, el grupo con *categoría*=10 habría cumplido la condición para grupos de la cláusula HAVING. El sexto paso es generar una fila de respuesta para cada grupo restante. La fila de respuesta correspondiente a cada grupo consiste en un subconjunto de las columnas de agrupación, y una o más columnas generadas mediante la aplicación de un operador de agregación. En este ejemplo cada fila de respuesta tiene una columna *categoría* y una columna *edadmín*, que se calcula aplicando MIN a los valores de la columna *edad* del grupo correspondiente. El resultado de este paso puede verse en la Figura 5.13.

<i>categoría</i>	<i>edad</i>
7	45,0
1	33,0
8	55,5
8	25,5
10	35,0
7	35,0
9	35,0
3	25,5
3	63,5
3	25,5

Figura 5.11 Tras la evaluación del paso 3

<i>categoría</i>	<i>edad</i>
1	33,0
3	25,5
3	25,5
3	63,5
7	45,0
7	35,0
8	55,5
8	25,5
9	35,0
10	35,0

Figura 5.12 Tras la evaluación del paso 4

<i>categoría</i>	<i>edadmín</i>
3	25,5
7	35,0
8	25,5

Figura 5.13 Resultado final de la evaluación del ejemplo

Ampliaciones de SQL:1999. Se han añadido dos nuevas funciones para conjuntos, **EVERY** y **ANY**. Cuando se emplean en la cláusula **HAVING**, la intuición básica de que la cláusula especifica una condición que debe satisfacer cada grupo, considerado en conjunto, no se modifica. Sin embargo, ahora la condición puede implicar pruebas para cada tupla del grupo, mientras que anteriormente se basaba de manera exclusiva en las funciones de agregación aplicadas al grupo de tuplas.

Si la consulta contiene **DISTINCT** en la cláusula **SELECT**, los duplicados se eliminan en un paso final adicional.

SQL:1999 ha introducido dos nuevas funciones para conjuntos, **EVERY** y **ANY**. Para ilustrarlas, se puede sustituir la cláusula **HAVING** del ejemplo por

```
HAVING COUNT (*) > 1 AND EVERY ( M.edad <= 60 )
```

El quinto paso de la evaluación conceptual es el afectado por la modificación de la cláusula **HAVING**. Considérese el resultado del cuarto paso, que puede verse en la Figura 5.12. La palabra clave **EVERY** exige que todas las filas de cada grupo satisfagan la condición adjunta para superar la condición para grupos. El grupo de la *categoría* 3 no cumple este criterio y se descarta; el resultado se muestra en la Figura 5.14.

Merece la pena contrastar la consulta anterior con la siguiente, en la que la condición para *edad* se halla en la cláusula **WHERE** en lugar de en la cláusula **HAVING**:

```

SELECT      M.categoría, MIN (M.edad) AS edadmín
FROM        Marineros M
WHERE       M.edad >= 18 AND M.edad <= 60
GROUP BY    M.categoría
HAVING     COUNT (*) > 1

```

Ahora, el resultado tras el tercer paso de la evaluación conceptual ya no contiene la fila con *edad* 63,5. No obstante, el grupo para la *categoría* 3 satisface la condición COUNT (*) > 1, ya que sigue teniendo dos filas, y cumple el criterio de condición de grupos aplicado en el quinto paso. El resultado final de esta consulta puede verse en la Figura 5.15.

<i>categoría</i>	<i>edadmín</i>
7	45,0
8	55,5

Figura 5.14 Resultado final de la consulta EVERY

<i>categoría</i>	<i>edadmín</i>
3	25,5
7	45,0
8	55,5

Figura 5.15 Resultado de la consulta alternativa

5.5.2 Más ejemplos de consultas de agregación

(C33) *Para cada barco rojo, averiguar el número de reservas realizadas.*

```

SELECT      B.idb, COUNT (*) AS númreservas
FROM        Barcos B, Reservas R
WHERE       R.idb = B.idb AND B.color = 'rojo'
GROUP BY    B.idb

```

Para los ejemplares *B1* y *R2*, la respuesta a esta consulta contiene las tuplas $\langle 102, 3 \rangle$ y $\langle 104, 2 \rangle$.

Obsérvese que esta versión de la consulta anterior es ilegal:

```

SELECT      B.idb, COUNT (*) AS númreservas
FROM        Barcos B, Reservas R
WHERE       R.idb = B.idb
GROUP BY    B.idb
HAVING     B.color = 'rojo'

```

Aunque la condición para grupos *B.color = 'rojo'* tiene un solo valor para cada grupo, ya que el atributo de agrupación *idb* es clave de Barcos (y, por tanto, determina *color*), SQL no permite esta consulta⁶. Sólo pueden aparecer en la cláusula HAVING las columnas que aparecen en la cláusula GROUP BY, a menos que aparezcan como argumentos de algún operador de agregación de la cláusula HAVING.

(C34) *Averiguar la edad media de los marineros de cada categoría que tenga, como mínimo, dos marineros.*

⁶Esta consulta se puede reescribir fácilmente para que sea legal en SQL:1999 empleando EVERY en la cláusula HAVING.

```

SELECT      M.categoría, AVG (M.edad) AS edadmed
FROM        Marineros M
GROUP BY    M.categoría
HAVING     COUNT (*) > 1

```

Tras identificar los grupos basados en la *categoría*, sólo se conservan los grupos con dos marineros como mínimo. La respuesta a esta consulta para el ejemplar *M3* puede verse en la Figura 5.16.

categoría	edadmed
3	44,5
7	40,0
8	40,5
10	25,5

Figura 5.16 Respuesta a C34

categoría	edadmed
3	45,5
7	40,0
8	40,5
10	35,0

Figura 5.17 Respuesta a C35

categoría	edadmed
3	45,5
7	40,0
8	40,5

Figura 5.18 Respuesta a C36

La siguiente formulación alternativa de la consulta C34 ilustra que la cláusula HAVING puede tener subconsultas anidadas, al igual que la cláusula WHERE. Obsérvese que se puede emplear *M.categoría* dentro de la consulta anidada de la cláusula HAVING, ya que tiene un solo valor para el grupo actual de marineros:

```

SELECT      M.categoría, AVG ( M.edad ) AS edadmed
FROM        Marineros M
GROUP BY    M.categoría
HAVING     1 < ( SELECT COUNT (*)
                  FROM  Marineros M2
                  WHERE M.categoría = M2.categoría )

```

(C35) Averiguar la edad media de los marineros con derecho a voto (es decir, con 18 años de edad como mínimo) para cada categoría que tenga, como mínimo, dos marineros.

```

SELECT      M.categoría, AVG ( M.edad ) AS edadmed
FROM        Marineros M
WHERE       M.edad >= 18
GROUP BY    M.categoría
HAVING     1 < ( SELECT COUNT (*)
                  FROM  Marineros M2
                  WHERE M.categoría = M2.categoría )

```

En esta variante de la consulta C34 primero se eliminan las tuplas con *edad* ≤ 18 y se agrupan las tuplas restantes por *categoría*. La subconsulta de la cláusula HAVING calcula para cada grupo el número de tuplas de Marineros (sin aplicar la selección *edad* ≤ 18) con el mismo valor de *categoría* que el grupo en cuestión. Si algún grupo tiene menos de dos marineros, se descarta. Para cada grupo restante se obtiene como resultado la edad media. La respuesta a esta consulta con el ejemplar *M3* puede verse en la Figura 5.17. Obsérvese que la respuesta es muy parecida a la de la consulta C34, con la única diferencia de que para el grupo de categoría 10 ahora se ignora al marinero de 16 años de edad al calcular la media.

(C36) Averiguar la edad media de los marineros que tienen derecho a voto (es decir, tienen, como mínimo, 18 años) para cada categoría que tiene, como mínimo, dos marineros así.

```
SELECT      M.categoría, AVG ( M.edad ) AS edadmed
FROM        Marineros M
WHERE       M. edad > 18
GROUP BY    M.categoría
HAVING     1 < ( SELECT COUNT (*)
                  FROM   Marineros M2
                  WHERE  M.categoría = M2.categoría AND M2.edad >= 18 )
```

Esta formulación de la consulta muestra su parecido con C35. La respuesta a C36 para el ejemplar *M3* puede verse en la Figura 5.18. Se diferencia de la respuesta a C35 en que no hay ninguna tupla para la categoría 10, ya que sólo hay una tupla de categoría 10 y $edad \geq 18$.

La consulta C36 es realmente muy parecida a C32, como muestra la siguiente formulación más sencilla:

```
SELECT      M.categoría, AVG ( M.edad ) AS edadmed
FROM        Marineros M
WHERE       M. edad > 18
GROUP BY    M.categoría
HAVING     COUNT (*) > 1
```

Esta formulación de C36 se aprovecha del hecho de que la cláusula **WHERE** se aplica antes de llevar a cabo la agrupación; por tanto, sólo quedan los marineros de $edad > 18$ cuando se agrupa. Resulta instructivo considerar otra manera más de escribir esta consulta:

```
SELECT Temp.categoría, Temp.edadmed
FROM   ( SELECT      M.categoría, AVG ( M.edad ) AS edadmed,
                COUNT (*) AS númcategorías
          FROM    Marineros M
          WHERE   M. edad > 18
          GROUP BY M.categoría ) AS Temp
WHERE Temp.númcategorías > 1
```

Esta alternativa trae a colación varios puntos interesantes. En primer lugar, la cláusula **FROM** también puede contener subconsultas anidadas de acuerdo con la norma de SQL⁷. En segundo lugar, la cláusula **HAVING** no es necesaria en absoluto. Cualquier consulta con cláusulas **HAVING** se puede reescribir sin ellas, pero muchas consultas resultan más sencillas de expresar con la cláusula **HAVING**. Finalmente, cuando aparecen subconsultas en la cláusula **FROM**, es necesario emplear la palabra clave **AS** para darles nombre (ya que, en caso contrario, no podrían expresar, por ejemplo, la condición *Temp.númcategorías > 1*).

(C37) Averiguar las categorías para las que la edad media de los marineros es mínima.

Esta consulta se emplea para ilustrar que las operaciones de agregación no se pueden anidar. Se podría considerar escribirla de la manera siguiente:

⁷No todos los sistemas comerciales de bases de datos soportan actualmente consultas anidadas en la cláusula **FROM**.

El modelo relacional y SQL. Los valores nulos no forman parte del modelo relacional básico. Al igual que el tratamiento de las tablas como multiconjuntos de tuplas en SQL, se trata de una extensión del modelo básico.

```
SELECT      M.categoría
FROM        Marineros M
WHERE       AVG (M.edad) = ( SELECT    MIN (AVG (M2.edad))
                           FROM      Marineros M2
                           GROUP BY M2.categoría )
```

Una pequeña reflexión muestra que esta consulta no funcionaría aunque la expresión `MIN (AVG (M2.edad))` estuviera permitida, la cual es ilegal. En la consulta anidada, `Marineros` se divide en grupos por categorías y la edad media se calcula para cada valor de la categoría. Para cada grupo, la aplicación de `MIN` a este valor medio de la edad del grupo devuelve el mismo valor. A continuación se ofrece una versión correcta de esta consulta. Fundamentalmente calcula una tabla temporal que contiene la edad media de cada valor de categoría y luego halla la(s) categoría(s) para las que esa edad media es mínima.

```
SELECT Temp.categoría, Temp.edadmed
FROM   ( SELECT   M.categoría, AVG (M.edad) AS edadmed,
                FROM     Marineros M
                GROUP BY M.categoría ) AS Temp
WHERE  Temp.edadmed = ( SELECT MIN (Temp.edadmed) FROM Temp )
```

La respuesta a esta consulta con el ejemplar *M3* es $\langle 10, 25, 5 \rangle$.

Como ejercicio, considérese si la consulta siguiente calcula la misma respuesta.

```
SELECT      Temp.categoría, MIN ( Temp.edadmed )
FROM        ( SELECT      M.categoría, AVG (M.edad) AS edadmed,
                FROM      Marineros M
                GROUP BY M.categoría ) AS Temp
GROUP BY Temp.categoría
```

5.6 VALORES NULOS

Hasta ahora se ha asumido que los valores de las columnas para una fila dada son siempre conocidos. En la práctica, los valores de las columnas pueden ser **desconocidos**. Por ejemplo, cuando un marinero, por ejemplo Dani, se hace socio de un club náutico, puede que no tenga todavía categoría asignada. Dado que la definición de la tabla `Marineros` tiene una columna `categoría`, ¿qué fila hay que insertar para Dani? Lo que hace falta es un valor especial que denote *desconocido*. Supóngase que la definición de la tabla `Marineros` se modifica para incluir la columna *segundo-apellido*. Sin embargo, sólo las personas de origen hispano tienen segundo apellido. Para las personas de otras nacionalidades la columna *segundo apellido* es *inaplicable*. Una vez más, ¿qué valor se incluye en esa columna para la fila que representa a Dani?

SQL ofrece un valor especial para las columnas denominado *null* (nulo) para emplearlo en estas situaciones. Se emplea *null* cuando el valor de la columna es *desconocido* o *inaplicable*.

Empleando la definición de la tabla *Marineros* se puede introducir la fila $\langle 98, Dani, null, 39 \rangle$ para representar a Dani. La presencia de valores *null* complica muchas situaciones y en este apartado se considera su impacto sobre SQL.

5.6.1 Comparaciones que emplean valores nulos

Considérese una comparación como *categoría* = 8. Si se aplica a la fila de Dani, ¿esta condición es verdadera o falsa? Dado que la categoría de Dani es desconocida, resulta razonable decir que esta consideración debería tomar el valor *desconocido*. De hecho, éste es el caso para las comparaciones *categoría* > 8 y también para *categoría* < 8. Acaso de manera menos obvia, si se comparan dos valores *null* mediante <, >, =, etcétera, el resultado siempre es *desconocido*. Por ejemplo, si se tiene *null* en dos filas distintas de la relación *Marineros*, cualquier comparación devuelve *desconocido*.

SQL también ofrece el operador de comparación especial *IS NULL* para comprobar si el valor de alguna columna es *null*; por ejemplo, se puede decir que *categoría IS NULL*, que tomaría el valor *verdadero* para la fila que representa a Dani. También se puede decir que *categoría IS NOT NULL*, que tomaría el valor *falso* para la fila de Dani.

5.6.2 Las conectivas lógicas AND, OR y NOT

Considérense ahora expresiones booleanas como *categoría* = 8 *OR* *edad* < 40 y *categoría* = 8 *AND* *edad* < 40. Tomando nuevamente en consideración la fila de Dani, como *edad* < 40, la primera expresión toma el valor de verdadera independientemente del valor de *categoría*, pero ¿qué ocurre con la segunda? Sólo se puede decir *desconocido*.

Pero este ejemplo destaca un problema importante —una vez se tienen valores *null* hay que definir los operadores lógicos AND, OR y NOT mediante una lógica *de tres valores* en la que las expresiones toman el valor de *verdadero*, *falso* o *desconocido*—. Se amplían las interpretaciones habituales de AND, OR y NOT para que cubran el caso en el que uno de los argumentos es *desconocido* de la manera siguiente. La expresión NOT *desconocido* se define como *desconocido*. OR para dos argumentos adopta el valor *verdadero* si cualquiera de los dos tiene el valor de *verdadero* y como *desconocido* si uno de los argumentos toma el valor *falso* y el otro toma el valor *desconocido*. (Si los dos argumentos son *falsos*, por supuesto, OR toma el valor *falso*.) AND para dos argumentos adopta el valor *falso* si cualquiera de los argumentos es *falso*, y *desconocido* si uno de ellos toma el valor *desconocido* y el otro toma el valor *verdadero* o *desconocido*. (Si los dos argumentos son *verdadero*, AND toma el valor *verdadero*.)

5.6.3 Consecuencias para las estructuras de SQL

Las expresiones booleanas aparecen en muchos contextos de SQL, y el efecto de los valores *null* se debe reconocer. Por ejemplo, la condición de la cláusula WHERE elimina filas (del producto cartesiano de tablas llamadas en la cláusula FROM) para las cuales la condición no toma el valor *verdadero*. Por tanto, en presencia de valores *null*, cualquier fila que tome el valor *falso* o *desconocido* se elimina. La eliminación de filas que toman el valor *desconocido* tiene un

efecto sutil pero significativo en las consultas, especialmente en las consultas anidadas que implican a `EXISTS` o a `UNIQUE`.

Otro problema con la presencia de valores `null` es la definición de la consideración como *duplicadas* de dos filas del mismo ejemplar de una relación dada. La definición de SQL es que dos filas están duplicadas si las columnas correspondientes son iguales o contienen valores `null`. Compárese esta definición con el hecho de que, si se comparan dos valores `null` con `=`, el resultado es *desconocido*. En el contexto de los duplicados, esta comparación se trata de manera implícita como *verdadera*, lo que constituye una anomalía.

Como cabía esperar, todas las operaciones aritméticas `+`, `-`, `*` y `/` devuelven `null` si uno de sus argumentos es `null`. Sin embargo, los valores `null` pueden provocar comportamientos inesperados de las operaciones de agregación. `COUNT(*)` trata los valores `null` igual que a los demás valores; es decir, los cuenta. Todas las demás operaciones de agregación (`COUNT`, `SUM`, `AVG`, `MIN`, `MAX` y las variaciones con `DISTINCT`) se limitan a descartar los valores `null` —así, `SUM` no puede comprenderse sólo como la suma de todos los valores del (multi)conjunto de valores al que se aplica; también se debe tener en cuenta el paso previo de descarte de los valores `null`—. Como caso especial, si uno de estos operadores —que no sea `COUNT`— se aplica sólo a valores `null`, el resultado es también `null`.

5.6.4 Reuniones externas

SQL soporta algunas variedades interesantes de la operación reunión que aprovechan los valores `null`, las denominadas **reuniones externas**. Considérese la reunión de dos tablas, por ejemplo, `Marineros` \bowtie_c `Reservas`. Las tuplas de `Marineros` que no coinciden con ninguna fila de `Reservas` según la condición de reunión c no aparecen en el resultado. En las reuniones externas, por otro lado, las filas de `Marineros` sin filas correspondientes en `Reservas` aparecen en el resultado exactamente una vez y con las columnas heredadas de `Reservas` asignadas a valores `null`.

De hecho, hay diversas variedades de la idea de reunión externa. En las **reuniones externas por la izquierda** las filas de `Marineros` sin fila correspondiente de `Reservas` aparecen en el resultado, pero no al contrario. En las **reuniones externas por la derecha** las filas de `Reservas` sin fila correspondiente de `Marineros` aparecen en el resultado, pero no al contrario. En las **reuniones externas completas** tanto las filas de `Marineros` como las de `Reservas` sin fila correspondiente aparecen en el resultado. (Por supuesto, las filas con correspondencias aparecen en el resultado en todas las variedades, igual que en las reuniones habituales, a veces denominadas *reuniones internas*, presentadas en el Capítulo 4.)

SQL permite especificar el tipo de reunión deseado en la cláusula `FROM`. Por ejemplo, la siguiente consulta muestra pares $\langle idm, idb \rangle$ correspondientes a marineros y a los barcos que han reservado:

```
SELECT M.idm, R.idb
  FROM Marineros M NATURAL LEFT OUTER JOIN Reservas R
```

La palabra clave `NATURAL` especifica que la condición de reunión es la igualdad para todos los atributos comunes (en este ejemplo, `idm`) y no se exige la cláusula `WHERE` (a menos que se desee especificar condiciones adicionales que no afecten a la reunión). Para los ejemplares de

Marineros y de Reservas mostrados en la Figura 5.6, esta consulta calcula el resultado que puede verse en las Figuras 5.6 y 5.7.

<i>idm</i>	<i>idb</i>
22	101
31	<i>null</i>
58	103

Figura 5.19 Reunión externa por la izquierda de *R3* y *M4*

5.6.5 Desactivación de los valores nulos

Se pueden impedir los valores *null* especificando NOT NULL como parte de la definición del campo; por ejemplo, *nombrem* CHAR(20) NOT NULL. Además, no se permite que los campos de la clave primaria adopten valores *null*. Por tanto, hay una restricción NOT NULL implícita para todos los campos que aparecen en la restricción PRIMARY KEY.

Esta cobertura de los valores *null* dista mucho de estar completa. El lector interesado deberá consultar alguno de los muchos libros dedicados a SQL para obtener un tratamiento más detallado de este tema.

5.7 RESTRICCIONES DE INTEGRIDAD COMPLEJAS EN SQL

En este apartado se estudia la especificación de restricciones de integridad complejas que aprovechan toda la potencia de las consultas SQL. Las características estudiadas en este apartado complementan las características de las restricciones de integridad de SQL presentadas en el Capítulo 3.

5.7.1 Restricciones sobre una sola tabla

Se pueden especificar restricciones complejas para una sola tabla mediante **restricciones de tabla**, que tienen la forma CHECK *expresión-condicional*. Por ejemplo, para garantizar que la *categoría* sea un entero en el rango de 1 a 10, se puede emplear:

```
CREATE TABLE Marineros ( idm      INTEGER,
                         nombrem  CHAR(10),
                         categoría INTEGER,
                         edad      REAL,
                         PRIMARY KEY (idm),
                         CHECK ( categoría >= 1 AND categoría <= 10 ))
```

Para hacer que se cumpla la restricción de que no se puede reservar el barco Intrépido se puede utilizar:

```

CREATE TABLE Reservas (
    idm      INTEGER,
    idb      INTEGER,
    fecha    DATE,
    FOREIGN KEY (idm) REFERENCES Marineros
    FOREIGN KEY (idb) REFERENCES Barcos
    CONSTRAINT noResIntrépido
    CHECK ( 'Intrépido' <>
            ( SELECT B.nombreb
              FROM   Barcos B
              WHERE  B.idb = Reservas.idb )))

```

Cuando se inserta alguna fila en Reservas o se modifica alguna fila ya existente, se evalúa la *expresión condicional* de la restricción CHECK. Si adopta el valor **falso**, se rechaza la orden.

5.7.2 Restricciones de dominio y tipos distintos

Los usuarios pueden definir nuevos dominios mediante la instrucción CREATE DOMAIN, que utiliza restricciones CHECK.

```

CREATE DOMAIN valcategoria INTEGER DEFAULT 1
                           CHECK ( VALUE >= 1 AND VALUE <= 10 )

```

INTEGER es el tipo subyacente, o *fuente*, del dominio valcategoria, y todos los valores de valcategoria deben ser de este tipo. Los valores de valcategoria quedan aún más restringidos por el empleo de la restricción CHECK; para definir esa restricción se utiliza la palabra clave VALUE para hacer referencia a los valores del dominio. Mediante esta característica se pueden restringir los valores que pertenecen a un dominio aprovechando toda la potencia de las consultas SQL. Una vez definido el dominio se puede utilizar su nombre para restringir los valores de las columnas de la tabla; por ejemplo, se puede emplear la línea siguiente para la declaración del esquema:

```
        categoría     valcategoria
```

La palabra clave opcional DEFAULT se utiliza para asociar un valor predeterminado con el dominio. Si se utiliza el dominio valcategoria para una columna de alguna relación y no se introduce ningún valor para esa columna en la tupla insertada, se utiliza el valor predeterminado 1 asociado con valcategoria.

El apoyo de SQL al concepto de dominio queda limitado en un aspecto importante. Por ejemplo, se pueden definir dos dominios denominados IdMarinero e IdBarco, cada uno de los cuales emplea INTEGER como tipo subyacente. La intención es obligar a que falle siempre la comparación entre los valores de IdMarinero y los de IdBarco (ya que se extraen de dominios diferentes); sin embargo, dado que los dos tienen el mismo tipo base, INTEGER, la comparación tendrá éxito en SQL. Este problema se aborda en SQL:1999 mediante la introducción de **tipos distintos**:

```
CREATE TYPE tipocategoría AS INTEGER
```

Esta instrucción define un nuevo tipo *distinto* denominado tipocategoría, con INTEGER como tipo fuente. Los valores del tipo tipocategoría se pueden comparar entre sí, pero no

Los tipos distintos en SQL:1999. Muchos sistemas, como UDS de Informix y DB2 de IBM, ya soportan esta característica. Con su introducción se espera que se *abandone* el soporte de los dominios y, finalmente, se elimine en versiones futuras de la norma SQL. Realmente no es más que una parte de un amplio conjunto de características orientadas a objetos de SQL:1999, que se tratan en el Capítulo 15.

se pueden comparar con valores de otros tipos. En concreto, los valores de **tipocategoría** se tratan como diferentes de los valores del tipo fuente, **INTEGER** —no se pueden comparar ni combinar con enteros (es decir, añadir enteros a valores **tipocategoría**)—. Si se desea definir operaciones para el tipo nuevo, por ejemplo, una función *promedio*, hay que hacerlo de manera explícita; no se puede aprovechar ninguna de las operaciones existentes del tipo origen. En el Apartado 15.4.1 se estudiará la manera en que se pueden definir estas funciones.

5.7.3 Asertos: RI para varias tablas

Las restricciones de tabla se asocian con una sola tabla, aunque la expresión condicional de la cláusula **CHECK** pueda hacer referencia a otras tablas. Las restricciones de tabla *sólo* son necesarias si la tabla no está vacía. Por tanto, cuando una restricción implica a dos o más tablas, a veces el mecanismo de las restricciones de tabla resulta engorroso y no responde exactamente a lo deseado. Para tratar estas situaciones SQL soporta la creación de **asertos**, que son restricciones que no están asociadas con ninguna tabla en concreto.

Como ejemplo, supóngase que se desea hacer que se cumpla la restricción de que el número de barcos más el número de marineros debe ser menor de 100. (Esta condición puede exigirse, por ejemplo, para entrar en la categoría de club náutico “pequeño”.) Se puede probar con la siguiente restricción de tabla:

```
CREATE TABLE Marineros (
    idm      INTEGER,
    nombrem CHAR(10),
    categoría INTEGER,
    edad     REAL,
    PRIMARY KEY (idm),
    CHECK ( categoría >= 1 AND categoría <= 10 )
    CHECK ( ( SELECT COUNT (M.idm) FROM Marineros M )
            + ( SELECT COUNT (B.idb) FROM Barcos B )
            < 100 ))
```

Esta solución tiene dos inconvenientes. Está asociada con *Marineros*, aunque implica a *Barcos* de manera completamente simétrica. Lo que es más importante, si la tabla *Marineros* está vacía, esta restricción (debido a la semántica de las restricciones de tabla) siempre se cumple por definición, aunque haya más de 100 filas en *Barcos*. La especificación de esta restricción se puede ampliar para que compruebe que *Marineros* no esté vacía, pero ese enfoque se vuelve engorroso. La mejor solución es crear un aserto de la manera siguiente:

```
CREATE ASSERTION Clubpequeño
CHECK ( ( SELECT COUNT (M.idm) FROM Marineros M )
```

```
+ ( SELECT COUNT (B.idb) FROM Barcos B)
< 100 )
```

5.8 DISPARADORES Y BASES DE DATOS ACTIVAS

Los **disparadores** son procedimientos que el SGBD invoca automáticamente en respuesta a cambios concretos de la base de datos, que suele especificar el DBA. Las bases de datos con un conjunto de disparadores asociados se denominan **bases de datos activas**. La descripción de cada disparador contiene tres partes:

- **Evento:** una modificación de la base de datos que **activa** el disparador.
- **Condición:** una consulta o prueba que se ejecuta cuando se activa el disparador.
- **Acción:** un procedimiento que se ejecuta cuando se activa el disparador y su condición es verdadera.

Se puede considerar a los disparadores como “demonios” que controlan la base de datos y se ejecutan cuando se modifica la base de datos de modo que coincida con la especificación del *evento*. Los disparadores pueden ser activados por instrucciones de inserción, eliminación o actualización independientemente del usuario o de la aplicación que invocara la instrucción que lo activó; puede que los usuarios ni siquiera sean conscientes de que se ha ejecutado un disparador como efecto secundario de su programa.

La *condición* del disparador puede ser una instrucción verdadero/falso (por ejemplo, todos los sueldos de los empleados son menores de 100.000 €) o una consulta. La consulta se interpreta como *verdadera* si el conjunto de respuestas no está vacío y como *falsa* si no tiene ninguna respuesta. Si la condición adopta el valor verdadero, se ejecuta la acción asociada con el disparador.

La *acción* del disparador puede examinar las respuestas a la consulta de la condición del disparador, hacer referencia a los valores antiguos y nuevos de tuplas modificadas por la instrucción que activa el disparador, ejecutar consultas nuevas o realizar modificaciones de la base de datos. De hecho, la acción puede incluso ejecutar una serie de órdenes para la definición de datos (por ejemplo, crear tablas nuevas o modificar autorizaciones) u orientadas a las transacciones (por ejemplo, comprometer) o llamar a procedimientos del lenguaje anfitrión.

Un aspecto importante de los disparadores es el momento en que se ejecuta la acción en relación con la instrucción que ha activado el disparador. Por ejemplo, una instrucción que inserte registros en la tabla Alumnos puede activar un disparador que se emplee para conservar estadísticas sobre el número de alumnos menores de 18 años insertados de una vez por una instrucción de inserción típica. En función de lo que haga exactamente el disparador, puede que se desee que la acción se ejecute *antes* de que se apliquen las modificaciones a la tabla Alumnos o *después*. Los disparadores que inicialicen una variable empleada para contar el número de inserciones que cumplen la condición establecida deben ejecutarse antes, mientras que los disparadores que se ejecuten una vez por cada registro insertado que cumpla la condición establecida e incrementen la variable deben ejecutarse después de la inserción de cada registro (ya que puede que deseemos examinar los valores del nuevo registro para determinar la acción correspondiente).

5.8.1 Ejemplos de disparadores en SQL

Los ejemplos de la Figura 5.20, escritos empleando la sintaxis de Oracle Server para definir disparadores, ilustran los conceptos básicos subyacentes a los disparadores. (La sintaxis de SQL:1999 para estos disparadores es parecida; en breve se verá un ejemplo que emplea la sintaxis de SQL:1999.) El disparador denominado *inic_cuenta* inicializa una variable contadora antes de cada ejecución de la instrucción `INSERT` que añade tuplas a la relación `Alumnos`. El disparador denominado *aum_cuenta* incrementa el contador por cada tupla insertada que satisface la condición *edad < 18*.

```

CREATE TRIGGER inic_cuenta BEFORE INSERT ON Alumnos          /* Evento */
DECLARE
    cuenta INTEGER;
BEGIN
    cuenta := 0;
END

CREATE TRIGGER aum_cuenta AFTER INSERT ON Alumnos           /* Evento */
WHEN (new.edad < 18)      /* Condición; 'new' es la tupla recién insertada */
FOR EACH ROW
BEGIN      /* Acción; procedimiento con la sintaxis de PL/SQL de Oracle */
    cuenta := cuenta + 1;
END

```

Figura 5.20 Ejemplos que ilustran los disparadores

Uno de los disparadores de ejemplo de la Figura 5.20 se ejecuta antes que la instrucción de activación y el otro, después. También se pueden programar los disparadores para que se ejecuten *en lugar de* la instrucción de activación; o de manera *diferida*, al final de la transacción que contiene la instrucción de activación; o de manera *asíncrona*, como parte de una transacción diferente.

El ejemplo de la Figura 5.20 ilustra otro aspecto importante de la ejecución de los disparadores: los usuarios deben poder especificar si cada disparador se debe ejecutar una vez por registro modificado o una vez por instrucción de activación. Si la acción depende de cada registro modificado, por ejemplo, hay que examinar el campo *edad* del registro insertado de `Alumnos` para decidir si hay que incrementar la cuenta, se debe definir que el evento de disparo tenga lugar para cada registro modificado; para hacer esto se emplea la cláusula `FOR EACH ROW`. Estos disparadores se denominan **disparadores por filas**. Por otro lado, el disparador *inic_cuenta* sólo se ejecuta una vez por instrucción `INSERT`, independientemente del número de registros insertados, ya que se ha omitido la expresión `FOR EACH ROW`. Estos disparadores se denominan **disparadores por instrucciones**.

En la Figura 5.20 la palabra clave `new` hace referencia a la tupla recién insertada. Si se modificara alguna tupla ya existente se podrían utilizar las palabras clave `old` y `new` para hacer referencia a los valores anteriores y posteriores a la modificación. SQL:1999 también

permite que la acción del disparador haga referencia al *conjunto* de registros modificados, en vez de limitarse a un registro modificado cada vez. Por ejemplo, sería útil poder hacer referencia al conjunto de registros insertados de Alumnos en un disparador que se ejecute una vez tras la instrucción `INSERT`; se podría contar el número de registros insertados con `edad < 18` mediante una consulta SQL para ese conjunto. En la Figura 5.21 se muestra un disparador de ese tipo, que es una alternativa a los disparadores de la Figura 5.20.

La definición de la Figura 5.21 utiliza la sintaxis de SQL:1999 con objeto de ilustrar los parecidos y diferencias con respecto a la sintaxis empleada en un SGBD típico actual. La cláusula de la palabra clave `NEW TABLE` permite dar nombre de tabla (`TuplasInsertadas`) al conjunto de tuplas recién insertadas. La cláusula `FOR EACH STATEMENT` especifica un disparador por instrucciones y se puede omitir, ya que es el valor predeterminado. Esta definición no tiene cláusula `WHEN`; si se incluye una cláusula de este tipo, va detrás de la cláusula `FOR EACH STATEMENT`, justo antes de la especificación de la acción correspondiente.

El disparador se evalúa una vez por cada instrucción de SQL que inserte tuplas en `Alumnos`, e inserta una sola tupla en la tabla que contiene estadísticas de las modificaciones de las tablas de la base de datos. Los dos primeros campos de la tupla contienen constantes (que identifican a la tabla modificada, `Alumnos`, y el tipo de instrucción que la ha modificado, una instrucción `INSERT`), y el tercer campo es el número de tuplas de `Alumnos` insertadas con `edad < 18`. (El disparador de la Figura 5.20 sólo calcula el recuento; hace falta un disparador adicional para insertar la tupla correspondiente en la tabla de estadísticas.)

```
CREATE TRIGGER definir_cuenta AFTER INSERT ON Alumnos          /* Evento */
  REFERENCING NEW TABLE AS TuplasInsertadas
  FOR EACH STATEMENT
    INSERT           /* Acción */
      INTO TablaEstadísticas (TablaModificada, TipoModificación, Cuenta)
        SELECT 'Alumnos', 'Insert', COUNT *
        FROM TuplasInsertadas T
        WHERE T.edad < 18
```

Figura 5.21 Disparador orientado a conjuntos

5.9 DISEÑO DE BASES DE DATOS ACTIVAS

Los disparadores ofrecen un potente mecanismo para tratar las modificaciones de la base de datos, pero se deben emplear con precaución. El efecto de un conjunto de disparadores puede ser muy complejo, y el mantenimiento de una base de datos activa puede volverse muy difícil. A menudo, el empleo juicioso de las restricciones de integridad puede sustituir el de los disparadores.

5.9.1 ¿Por qué los disparadores pueden resultar difíciles de comprender?

En los sistemas de bases de datos activas, cuando el SGBD va a ejecutar alguna instrucción que modifique la base de datos, comprueba si esa instrucción activa algún disparador. En caso positivo, el SGBD procesa el disparador mediante la evaluación de su condición y, luego, (si la condición toma el valor verdadero) la ejecución de su acción.

Si alguna instrucción activa más de un disparador, el SGBD suele procesar todos, en un orden arbitrario. Es importante recordar que la ejecución de la acción de un disparador puede, a su vez, activar otro disparador. En especial, la ejecución de la acción de un disparador puede activar nuevamente ese mismo disparador; estos disparadores se denominan **disparadores recursivos**. La posibilidad de esas activaciones *en cadena* y el orden impredecible en que el SGBD procesa los disparadores activados pueden hacer difícil de comprender el efecto de los conjuntos de disparadores.

5.9.2 Restricciones y disparadores

Un empleo habitual de los disparadores es el mantenimiento de la consistencia de la base de datos y, en esos casos, siempre se debe tomar en consideración si el empleo de restricciones de integridad (por ejemplo, una restricción de clave externa) consigue los mismos objetivos. El significado de las restricciones no se define operativamente, a diferencia del efecto de los disparadores. Esta propiedad facilita la comprensión de las restricciones, y también da más oportunidades al SGBD para optimizar la ejecución. Las restricciones también evitan que los datos se vuelvan inconsistentes por *alguna* clase de instrucción, mientras que los disparadores los activa una clase concreta de instrucción (`INSERT`, `DELETE` o `UPDATE`). Una vez más, esta restricción hace que las restricciones sean más sencillas de comprender.

Por otro lado, los disparadores permiten conservar la integridad de la base de datos de maneras más flexibles, como ilustran los ejemplos siguientes.

- Supóngase que se tiene una tabla llamada Pedidos con los campos *idartículo*, *cantidad*, *idcliente* y *preciounitario*. Cuando un cliente formula un pedido, el valor de los tres primeros campos lo rellena el usuario (en este ejemplo, un comercial). El valor del cuarto campo se puede obtener de una tabla denominada Artículos, pero es importante incluirlo en la tabla Pedidos para tener un registro completo del pedido, por si el precio del artículo cambia posteriormente. Se puede definir un disparador que examine ese valor y lo incluya en el cuarto campo del registro recién insertado. Además de reducir el número de campos que el comercial tiene que llenar, el disparador elimina la posibilidad de que se introduzca un error que provoque un precio inconsistente en la tabla Pedidos.
- Siguiendo con este ejemplo, puede que se desee llevar a cabo alguna acción adicional cuando se reciba un pedido. Por ejemplo, si la compra se carga a una línea de crédito concedida por la empresa, puede que se desee comprobar si el coste total de la adquisición se halla dentro del límite de crédito. Se puede emplear un disparador para que haga esa comprobación; en realidad, incluso se puede utilizar una restricción `CHECK`. Sin embargo, el empleo de un disparador permite implementar políticas más sofisticadas pa-

ra el tratamiento de adquisiciones que superen el límite de crédito. Por ejemplo, puede que se permitan adquisiciones que superen el límite en un máximo del 10% si el cliente ha trabajado con la empresa un mínimo de un año, y añadir el cliente a una tabla de candidatos a aumentos del límite de crédito.

5.9.3 Otros usos de los disparadores

Muchos posibles usos de los disparadores van más allá del mantenimiento de la integridad. Los disparadores pueden alertar a los usuarios de eventos infrecuentes (como se refleja en las actualizaciones de la base de datos). Por ejemplo, puede que se desee comprobar si el cliente que formula un pedido ha realizado suficientes adquisiciones el mes anterior como para tener derecho a un descuento adicional; en caso positivo, se debe informar al comercial de que puede indicárselo al cliente y, posiblemente, generar ventas adicionales. Esta información se puede transmitir mediante un disparador que compruebe las adquisiciones recientes e imprima un mensaje si el cliente tiene derecho a ese descuento.

Los disparadores pueden generar un registro de los eventos para apoyar las auditorías y los controles de seguridad. Por ejemplo, cada vez que un cliente formula un pedido, se puede crear un registro con el identificador del cliente y su límite de crédito actual e insertarlo en una tabla con el historial de los clientes. El análisis posterior de esta tabla pudiera sugerir candidatos para un límite de crédito ampliado (por ejemplo, clientes que nunca han dejado de pagar a tiempo una factura y que se han quedado con menos del 10% de su crédito al menos tres veces durante el mes anterior).

Como ilustran los ejemplos del Apartado 5.8, se pueden emplear disparadores para reunir datos estadísticos de acceso a las tablas y de sus modificaciones. Algunos sistemas de bases de datos incluso emplean internamente los disparadores como base de la gestión de réplicas de las relaciones. Esta lista de usos posibles de los disparadores no es exhaustiva; por ejemplo, también se ha considerado el empleo de disparadores para la gestión de flujos de trabajo y para hacer que se cumplan las reglas de negocio.

5.10 PREGUNTAS DE REPASO

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

- ¿Cuáles son las partes de una consulta básica de SQL? ¿Son conjuntos o multiconjuntos las tablas de datos y de resultados de las consultas SQL? ¿Cómo se puede obtener un conjunto de tuplas como resultado de una consulta? (**Apartado 5.2**)
- ¿Qué son las variables de rango de SQL? ¿Cómo se puede poner nombre a las columnas de resultados de las consultas que se definen mediante expresiones aritméticas o de cadenas de caracteres? ¿Qué soporte ofrece SQL a la coincidencia estricta de patrones? (**Apartado 5.2**)
- ¿Qué operaciones ofrece SQL para los (multi)conjuntos de tuplas y cómo se pueden aprovechar para escribir consultas? (**Apartado 5.3**)
- ¿Qué son las consultas anidadas? ¿Qué es la *correlación* de las consultas anidadas? ¿Cómo se pueden utilizar los operadores IN, EXISTS, UNIQUE, ANY y ALL para escribir consultas

anidadas? ¿Por qué resultan útiles? Ilústrese la respuesta mostrando la manera de escribir el operador *division* en SQL. (**Apartado 5.4**)

- ¿Qué operadores de agregación soporta SQL? (**Apartado 5.5**)
- ¿Qué es la *agrupación*? ¿Existe un equivalente en el álgebra relacional? Explíquese esta característica y discútase la interacción de las cláusulas HAVING y WHERE. Mencíñese cualquier restricción que deban cumplir los campos que aparecen en la cláusula GROUP BY. (**Apartado 5.5.1**)
- ¿Qué son los valores *null*? ¿Están soportados en el modelo relacional, tal y como se describe en el Capítulo 3? ¿Cómo afectan al significado de las consultas? ¿Pueden contener valores *null* los campos de la clave principal de las tablas? (**Apartado 5.6**)
- ¿Qué tipos de restricciones de SQL se pueden especificar mediante el lenguaje de consulta? ¿Se pueden expresar las restricciones de clave principal mediante alguna de estos nuevos tipos de restricciones? En caso positivo, ¿por qué ofrece SQL una sintaxis diferente para la restricción de la clave primaria? (**Apartado 5.7**)
- ¿Qué son los *disparadores* y cuáles son las tres partes en que se dividen? ¿Cuáles son las diferencias entre los disparadores por filas y los disparadores por instrucciones? (**Apartado 5.8**)
- ¿Por qué los disparadores pueden resultar difíciles de comprender? Explíquense las diferencias entre los disparadores y las restricciones de integridad y descríbase cuándo se preferirán unos u otras. ¿Para qué se emplean los disparadores? (**Apartado 5.9**)

EJERCICIOS



Se dispone de material en línea para todos los ejercicios de este capítulo en la página Web del libro en

<http://www.cs.wisc.edu/~dbbook>

En esa página se incluyen secuencias de comandos para crear tablas para cada ejercicio para usarlas con Oracle, DB2 de IBM, SQL Server de Microsoft, Microsoft Access y MySQL.

Ejercicio 5.1 Considérense las relaciones siguientes:

```
Alumnos(numa: integer, nombrea: string, carrera: string, nivel: string, edad: integer)
Cursos(nombre: string, impartido_en: string, aula: string, idp: integer)
Matriculado(numa: integer, nombrec: string)
Profesores(idp: integer, nombrep: string, iddep: integer)
```

El significado de estas relaciones es evidente; por ejemplo, Matrículado tiene un registro para cada par alumno-curso tal que el alumno está matriculado en ese curso.

Escríbanse las consultas siguientes en SQL. No se permiten duplicados en ninguna de las respuestas.

1. Averiguar el nombre de todos los alumnos de tercer año (nivel = JR) que están matriculados en alguna clase impartida por El Profe.
2. Averiguar la edad del alumno de más edad que tiene un título de Historia o está matriculado en algún curso impartido por El Profe.

3. Averiguar el nombre de todos los cursos que se imparten en el aula A128 o tienen matriculados cinco o más alumnos.
4. Averiguar el nombre de todos los alumnos que están matriculados en dos cursos que se imparten al mismo tiempo.
5. Averiguar el nombre de los profesores que dan clase en todas las aulas en las que se imparte alguna clase.
6. Averiguar el nombre de los profesores para los que la matrícula total de las asignaturas que imparten es menor de cinco.
7. Para cada nivel, determinar el nivel y la edad media de los alumnos de ese nivel.
8. Para todos los niveles excepto el tercer año, determinar el nivel y la edad media de los alumnos de ese nivel.
9. Para cada profesor que sólo haya dado clase en el aula A128, determinar el nombre del profesor y el número total de cursos que ha dado.
10. Averiguar el nombre de los alumnos matriculados en mayor número de cursos.
11. Averiguar el nombre de los alumnos no matriculados en ningún curso.
12. Para cada valor de edad que aparezca en Alumnos, averiguar el valor de nivel que aparece más a menudo. Por ejemplo, si hay más alumnos de primer año con 18 de edad que alumnos de cuarto (SR), tercero (JR) o segundo año (SO) con esa edad, hay que determinar la pareja (18, FR).

Ejercicio 5.2 Considérese el esquema siguiente:

```
Proveedores(idp: integer, nombrp: string, domicilio: string)
Repuestos(idr: integer, nombrer: string, color: string)
Catálogo(idp: integer, idr: integer, coste: real)
```

La relación Catálogo muestra el precio que cobran los proveedores por los repuestos. Escribanse las consultas siguientes en SQL:

1. Averiguar el *nombrer* de los repuestos para los que hay algún proveedor.
2. Averiguar el *nombrp* de los proveedores que suministran todos los repuestos.
3. Averiguar el *nombrp* de los proveedores que suministran todos los repuestos rojos.
4. Averiguar el *nombrer* de los repuestos suministrados por el proveedor Repuestos Acme y por nadie más.
5. Averiguar el *idp* de los proveedores que cobran más por algún repuesto que el coste medio de ese repuesto (promediado sobre todos los proveedores que suministran ese repuesto).
6. Para cada repuesto, averiguar el *nombrp* del proveedor que cobra más por ese repuesto.
7. Averiguar el *idp* de los proveedores que sólo suministran repuestos rojos.
8. Averiguar el *idp* de los proveedores que suministran un repuesto rojo y otro verde.
9. Averiguar el *idp* de los proveedores que suministran un repuesto rojo o uno verde.
10. Para cada proveedor que sólo suministra repuestos verdes, determinar el nombre del proveedor y el número total de repuestos que suministra.
11. Para cada proveedor que suministra un repuesto verde y uno rojo, determinar el nombre y el precio del repuesto más caro que suministre.

Ejercicio 5.3 Las relaciones siguientes realizan el seguimiento de la información de vuelos de una línea aérea:

```
Vuelos(nuvu: integer, de: string, a: string, distancia: integer,
       salida: time, llegada: time, precio: real)
Aviones(ida: integer, nombrea: string, autonomía: integer)
Certificado(ide: integer, ida: integer)
Empleados(ide: integer, nombree: string, sueldo: integer)
```

Obsérvese que la relación Empleados describe a los pilotos y también a otros tipos de empleados; todos los pilotos están homologados para determinados tipos de aviones, y sólo los pilotos están homologados para volar. Escríbase cada una de las consultas siguientes en SQL. (*Hay más consultas que utilizan el mismo esquema en los ejercicios del Capítulo 4.*)

1. Averiguar el nombre de los aviones tales que todos los pilotos homologados para operar con ellos tienen sueldos superiores a 80.000 €.
2. Para cada piloto homologado para más de tres aviones, averiguar el *ide* y la *autonomía* máxima de los aviones para los que está homologado.
3. Averiguar el nombre de los pilotos cuyo *sueldo* es menor que el precio de la ruta más barata de Las Palmas a Hamburgo.
4. Para todos los aviones con *autonomía* superior a 2.000 kilómetros, averiguar el nombre del avión y el sueldo medio de todos los pilotos homologados para ese modelo.
5. Averiguar el nombre de los pilotos homologados para algún avión de Airbus.
6. Averiguar el *ida* de todos los aviones que se pueden utilizar en rutas de Las Palmas a Cracovia.
7. Identificar las rutas que pueden pilotar todos los pilotos que ganan más de 100.000 €.
8. Determinar el *nombree* de los pilotos que pueden operar con aviones de *autonomía* superior a 5.000 kilómetros pero que no están homologados para ningún avión de Airbus.
9. Un cliente desea viajar de Málaga a Nancy con no más de dos transbordos. Indíquese la selección de horas de salida desde Málaga si el cliente desea llegar a Nancy hacia las seis de la tarde.
10. Calcúlese la diferencia entre el sueldo medio de los pilotos y el sueldo medio de todos los empleados (incluidos los pilotos).
11. Escríbase el nombre y el sueldo de todos los empleados que no son pilotos cuyo sueldo es superior al sueldo medio de los pilotos.
12. Determinar el nombre de los empleados que sólo están homologados para aviones con autonomía superior a 2.000 kilómetros.
13. Determinar el nombre de los empleados que sólo están homologados para aviones con autonomía superior a 2.000 kilómetros, pero, como mínimo, para dos aviones.
14. Determinar el nombre de los empleados que sólo están homologados para aviones con autonomía superior a 2.000 kilómetros y para algún avión de Airbus.

Ejercicio 5.4 Considérese el siguiente esquema relacional. Cada empleado puede trabajar en más de un departamento; el campo *tiempo_parcial* de la relación Trabaja muestra el porcentaje de tiempo que cada empleado trabaja en un departamento dado.

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

Escribir las consultas siguientes en SQL:

1. Determinar el nombre y la edad de cada empleado que trabaje tanto en el departamento de Ferretería como en el de Software.
2. Para cada departamento con más de 20 empleados equivalentes a tiempo completo (es decir, en el que los empleados a tiempo parcial y a tiempo completo suman, como mínimo, el equivalente a esos empleados a tiempo completo), determinar el *idd* junto con el número de empleados que trabajan en ese departamento.
3. Determinar el nombre de cada empleado cuyo sueldo supere el presupuesto de todos los departamentos en los que trabaja.
4. Averiguar el *idencargado* de los encargados que sólo dirigen departamentos con presupuestos mayores de 1 millón de euros.

<i>idm</i>	<i>nombrem</i>	<i>categoría</i>	<i>edad</i>
18	jiménez	3	30,0
41	jaraiz	6	56,0
22	ahab	7	44,0
63	moby	<i>null</i>	15,0

Figura 5.22 Un ejemplar de Marineros

5. Averiguar el *nombree* de los encargados que dirigen los departamentos de mayor presupuesto.
6. Supóngase que si un encargado dirige más de un departamento entonces *controla* la suma de los presupuestos de esos departamentos. Averiguar el *idencargado* de los encargados que controlan más de 5 millones de euros.
7. Averiguar el *idencargado* de los encargados que controlan los mayores importes.
8. Averiguar el *nombree* de los encargados que sólo dirigen departamentos con presupuestos de más de 1 millón de euros pero, como mínimo, un departamento con presupuesto menor de 5 millones de euros.

Ejercicio 5.5 Considérese el ejemplar de la relación Marineros que puede verse en la Figura 5.22.

1. Escríbanse consultas de SQL para calcular la categoría media, empleando **AVG**; la suma de categorías, utilizando **SUM**; y el número de categorías, usando **COUNT**.
2. Si se divide la suma recién calculada por la cuenta, ¿el resultado será igual que el promedio? ¿Cómo cambiaría la respuesta si este procedimiento se llevara a cabo para el campo *edad* en lugar de para *categoría*?
3. Considérese la consulta siguiente: *Averiguar el nombre de los marineros de categoría superior a todos los marineros de edad < 21*. Las dos consultas de SQL siguientes intentan obtener la respuesta a esta pregunta. ¿Calculan las dos el resultado buscado? En caso de no hacerlo, explíquese el motivo. ¿En qué condiciones calcularían el mismo resultado?

```

SELECT M.nombrem
FROM   Marineros M
WHERE  NOT EXISTS ( SELECT *
                     FROM   Marineros M2
                     WHERE  M2.edad < 21
                            AND M.categoría <= M2.categoría )

SELECT *
FROM   Marineros M
WHERE  M.categoría > ANY ( SELECT M2.categoría
                            FROM   Marineros M2
                            WHERE  M2.edad < 21 )

```

4. Considérese el ejemplar de Marineros de la Figura 5.22. Definamos el ejemplar M1 de Marineros como consistente en las dos primeras tuplas, el ejemplar M2 como consistente en las dos últimas tuplas y M como el ejemplar dada.
 - (a) Mostrar la reunión externa por la izquierda de M consigo misma, siendo la condición de reunión *idm=idm*.
 - (b) Mostrar la reunión externa por la derecha de M consigo misma, siendo la condición de reunión *idm=idm*.
 - (c) Mostrar la reunión externa completa de M consigo misma, siendo la condición de reunión *idm=idm*.

- (d) Mostrar la reunión externa por la izquierda de M1 con M2, siendo la condición de reunión $idm=idm$.
- (e) Mostrar la reunión externa por la derecha de M1 con M2, siendo la condición de reunión $idm=idm$.
- (f) Mostrar la reunión externa completa de M1 con M2, siendo la condición de reunión $idm=idm$.

Ejercicio 5.6 Considérese este esquema relacional y respóndase brevemente a las preguntas siguientes:

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

1. Defínase una restricción de tabla para Emp que garantice que todos los empleados ganan, como mínimo 10.000 €.
2. Defínase una restricción de tabla para Dep que garantice que todos los encargados tienen $edad > 30$.
3. Defínase un aserto para Dep que garantice que todos los empleados tengan $edad > 30$. Compárese este aserto con la restricción de tabla equivalente. Explíquese cuál es mejor.
4. Escríbanse instrucciones de SQL para eliminar toda la información sobre los empleados superen los del encargado de uno o varios de los departamentos en los que trabajan. Hay que asegurarse de que se satisfacen todas las restricciones de integridad relevantes se cumplen tras las actualizaciones.

Ejercicio 5.7 Considérense las relaciones siguientes:

```
Alumnos(numa: integer, nombrea: string, carrera: string,
         nivel: string, edad: integer)
Cursos(nombre: string, impartido_en: time, aula: string, idp: integer)
Matriculado(numa: integer, nombrec: string)
Profesores(idp: integer, nombrep: string, iddep: integer)
```

El significado de estas relaciones es evidente; por ejemplo, Matriculado tiene un registro por cada par alumno-curso tal que el alumno está matriculado en ese curso.

1. Escríbanse las instrucciones de SQL necesarias para crear estas relaciones, incluidas las versiones adecuadas de todas las restricciones de integridad de clave principal y de clave externa.
2. Exprésense cada una de las restricciones de integridad siguientes en SQL, a menos que alguna esté implícita en la restricción de clave primaria y de clave externa; en ese caso, explíquese la manera en que está implícita. Si la restricción no se puede expresar en SQL, indíquese. Para cada restricción, indíquese qué operaciones (inserciones, eliminaciones y actualizaciones de relaciones concretas) se deben controlar para hacer que se cumpla esa restricción.
 - (a) Todos los cursos tienen una matrícula mínima de 5 alumnos y máxima de 30.
 - (b) Como mínimo se imparte un curso en cada aula.
 - (c) Todos los profesores deben impartir, como mínimo, dos asignaturas.
 - (d) Sólo los profesores del departamento con $iddep=33$ imparten más de tres asignaturas.
 - (e) Todos los alumnos deben estar matriculados en la asignatura denominada Mat101.
 - (f) El aula en la que se imparte el curso programado en primer lugar (es decir, el curso con el valor mínimo de $impartido_en$) no debe ser la misma en que se imparte el curso programado en último lugar.
 - (g) No se pueden impartir dos cursos en la misma aula al mismo tiempo.
 - (h) El departamento con más profesores debe tener menos del doble que el departamento con menos profesores.
 - (i) Ningún departamento puede tener más de 10 profesores.

- (j) Cada alumno no puede añadir más de dos asignaturas a la vez (es decir, en una sola actualización).
- (k) El número de alumnos de cuarto curso de Informática debe ser mayor que el de alumnos de cuarto curso de Matemáticas.
- (l) El número de asignaturas diferentes en las que hay matriculados alumnos de cuarto curso de Informática es mayor que el de asignaturas diferentes en que hay matriculados alumnos de cuarto curso de Matemáticas.
- (m) La matrícula total de las asignaturas impartidas por los profesores del departamento con $iddep=33$ es mayor que el número de alumnos de cuarto curso de Matemáticas.
- (n) Debe haber, como mínimo, un alumno de cuarto curso de Informática si es que hay algún alumno.
- (o) Los profesores de departamentos diferentes no pueden dar clase en la misma aula.

Ejercicio 5.8 Discútanse los puntos fuertes y débiles del mecanismo de los disparadores. Compárense los disparadores con otras restricciones de integridad soportadas por SQL.

Ejercicio 5.9 Considérese el siguiente esquema relacional. Cada empleado puede trabajar en más de un departamento; el campo *tiempo_parcial* de la relación Trabaja muestra el porcentaje de tiempo que cada empleado trabaja en los diferentes departamentos.

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

Escríbanse restricciones de integridad en SQL-92 (de dominio, de clave, de clave externa o CHECK; o asertos) o disparadores en SQL:1999 para garantizar cada uno de los requisitos siguientes, considerados de manera independiente.

1. Los empleados deben ganar un sueldo mínimo de 1.000 €.
2. Todos los encargados deben ser también empleados.
3. El porcentaje total de nombramientos para cada empleado debe ser inferior al 100%.
4. Los encargados siempre deben tener un sueldo más elevado que cualquier empleado que dirijan.
5. Siempre que se conceda un aumento a un empleado se debe incrementar el sueldo del encargado en el mismo importe, como mínimo.
6. Siempre que se conceda un aumento a un empleado se debe incrementar el sueldo del encargado para que sea, como mínimo, igual. Además, siempre que se conceda un aumento a un empleado se debe incrementar el presupuesto del departamento para que sea mayor que la suma de los sueldos de todos los empleados del departamento.

EJERCICIOS BASADOS EN PROYECTOS

Ejercicio 5.10 Identifíquese el subconjunto de consultas de SQL soportadas en Minibase.

NOTAS BIBLIOGRÁFICAS

La versión original de SQL se desarrolló como lenguaje de consulta para el proyecto System R de IBM, y se puede seguir su primer desarrollo en [63, 94]. Desde entonces, SQL se ha transformado en el lenguaje relacional de consultas más utilizado y su desarrollo se halla sometido actualmente a un proceso internacional de normalización.

Melton y Simon presentan un tratamiento muy legible y completo de SQL-92 en Melton [326], y las características principales de SQL:1999 se tratan en [327]. Se remite a los lectores a estos libros si desean



un tratamiento autorizado de SQL. [155] presenta un breve resumen de la norma SQL:1999. Date ofrece una penetrante crítica de SQL en [134]. Aunque parte de los problemas se han abordado en SQL-92 y en revisiones posteriores, otros continúan. En [340] se presenta una semántica formal para un gran subconjunto de consultas de SQL. SQL:1999 es la norma actual de la Organización Internacional para la Normalización (International Organization for Standardization, ISO) y del Instituto Nacional Americano de Normalización (American National Standards Institute, ANSI). Melton es el editor de la norma ANSI e ISO SQL:1999, documento ANSI/ISO/IEC 9075-1:1999. El documento ISO correspondiente es ISO/IEC 9075-1:1999. Un sucesor, planeado para 2003, evolucionará a partir de SQL:1999. SQL:2003 se halla cercano a la ratificación (a junio de 2002). Los borradores de las deliberaciones sobre SQL:2003 están disponibles en el siguiente URL:

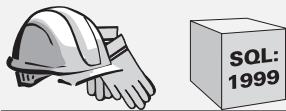
<ftp://sqlstandards.org/SC32/>

[471] contiene un conjunto de trabajos que tratan el campo de las bases de datos activas. [485] incluye una buena introducción en profundidad a las reglas activas, que trata los aspectos de la semántica, las aplicaciones y el diseño. [161] analiza las ampliaciones de SQL para la especificación de las comprobaciones de las restricciones de integridad mediante disparadores. [72] también estudia un mecanismo procedimental, denominado *alertador*, para el control de bases de datos. [117] es un trabajo reciente que sugiere la manera en que se pueden incorporar disparadores a las ampliaciones de SQL. Entre los prototipos influyentes de bases de datos están Ariel [240], HiPAC [321], ODE [9], Postgres [439] RDL [415], y Sentinel [22]. [91] compara diversas arquitecturas para los sistemas de bases de datos activas.

[19] considera las condiciones en las que un conjunto de reglas activas tiene el mismo comportamiento, independientemente del orden de evaluación. La semántica de las bases de datos activas se estudia también en [187] y en [483]. El diseño y administración de sistemas complejos de bases de datos se trata en [40, 150]. [88] estudia la gestión de las reglas mediante Chimera, un lenguaje y modelo de datos para los sistemas de bases de datos activas.

PARTE II

DESARROLLO DE APLICACIONES



6

DESARROLLO DE APLICACIONES DE BASE DE DATOS

- ¿Cómo se conectan las aplicaciones a un SGBD?
- ¿Cómo pueden manipular las aplicaciones datos obtenidos de un SGBD?
- ¿Cómo pueden modificar las aplicaciones datos en un SGBD?
- ¿Qué son los cursosres?
- ¿Qué es JDBC y cómo se utiliza?
- ¿Qué es SQLJ y cómo se utiliza?
- ¿Qué son los procedimientos almacenados?
- **Conceptos clave:** SQL incorporado, SQL dinámico, cursosres; JDBC, conexiones, controladores, ResultSets, java.sql, SQLJ; procedimientos almacenados, SQL/PSM.

Se beneficia más el que sirve mejor.

— Lema de Rotary International

En el Capítulo 5 se examinó un amplio rango de consultas SQL, tratando SQL como un lenguaje independiente por derecho propio. Un SGBD relacional soporta una interfaz *SQL interactiva*, y los usuarios pueden introducir comandos SQL directamente. Este enfoque simple está bien siempre que la tarea a realizar pueda llevarse a cabo enteramente con comandos SQL. En la práctica a menudo aparecen situaciones en las que es necesaria la flexibilidad de un lenguaje de programación de propósito general además de las características de manipulación de datos proporcionadas por SQL. Por ejemplo, es posible tener que integrar una aplicación de base de datos con una interfaz gráfica de usuario o con otras aplicaciones existentes.

Las aplicaciones que confían en el SGBD para gestionar datos se ejecutan como procesos separados que se conectan con el SGBD para interactuar con él. Una vez que se establece una conexión, se pueden utilizar comandos SQL para insertar, eliminar y modificar datos. Se

pueden utilizar consultas SQL para recuperar los datos deseados, pero es necesario sortear la distinta forma en que un sistema de base de datos ve los datos respecto de como los ve un programa de aplicación en un lenguaje como Java o C: el resultado de una consulta de base de datos es un conjunto (o multiconjunto) de registros, pero Java no tiene el tipo de datos conjunto o multiconjunto. Este desajuste se resuelve mediante construcciones SQL adicionales que permiten a las aplicaciones obtener un controlador sobre una colección e iterar sobre los registros uno a uno.

SQL incorporado, SQL dinámico y los cursores se introducen en el Apartado 6.1. SQL incorporado permite acceder a datos utilizando consultas SQL estáticas en el código de una aplicación (Apartado 6.1.1); con SQL dinámico se pueden crear las consultas en tiempo de ejecución (Apartado 6.1.3). Los cursores permiten acceder a respuestas de consultas como conjuntos desde lenguajes de programación que no tienen soporte para los valores de tipo conjunto (Apartado 6.1.2).

La aparición de Java como un lenguaje popular de desarrollo de aplicaciones, especialmente para aplicaciones de Internet, ha hecho que el acceso a un SGBD desde código Java sea un tema particularmente importante. El Apartado 6.4 describe JDBC, una interfaz de programación que permite ejecutar consultas SQL desde un programa Java y utilizar los resultados en el programa. JDBC proporciona mayor portabilidad que SQL incorporado o SQL dinámico, y ofrece la posibilidad de conectar con varios SGBD sin recomilar el código. El Apartado 6.4 describe SQLJ, que hace lo mismo para consultas SQL estáticas, pero que es más fácil de programar que Java con JDBC.

A menudo resulta útil ejecutar código de una aplicación en el servidor de la base de datos, en lugar de obtener los datos y ejecutar la lógica de la aplicación en un proceso separado. El Apartado 6.5 describe los procedimientos almacenados, que permiten que la lógica de las aplicaciones se almacene y ejecute en el servidor de base de datos. Se concluye el capítulo con el caso de estudio B&N en el Apartado 6.6.

Cuando se escriben aplicaciones de base de datos, también debe tenerse en cuenta que habitualmente muchos programas de aplicación se ejecutan concurrentemente. El concepto de transacción, introducido en el Capítulo 1, se utiliza para encapsular los efectos de una aplicación sobre la base de datos. Una aplicación puede seleccionar ciertas propiedades de la transacción por medio de comandos SQL para controlar el grado al que se expone a los cambios de otras aplicaciones que se ejecuten concurrentemente. El concepto de transacción se menciona en diversos puntos de este capítulo y, en particular, trata los aspectos relacionados con transacciones de JDBC. El estudio completo de las propiedades de las transacciones y el soporte de SQL para transacciones se realiza en el Capítulo 8.

Los ejemplos que aparecen en este capítulo están disponibles en inglés en:

<http://www.cs.wisc.edu/~dbbook>

6.1 ACCESO A BASES DE DATOS DESDE APLICACIONES

En este apartado se verá cómo se pueden ejecutar comandos SQL desde un programa en un **lenguaje anfitrión** como C o Java. El uso de comandos SQL desde un programa anfitrión se denomina **SQL incorporado**. Los detalles de SQL incorporado también dependen del



lenguaje anfitrión. Aunque las características similares de diversos lenguajes anfitriones están soportadas, la sintaxis puede variar.

Primero se tratarán los conceptos básicos de SQL incorporado con consultas SQL estáticas en el Apartado 6.1.1. Después se introducirán los cursos en el Apartado 6.1.2. Finalmente, en el Apartado 6.1.3 se describe SQL dinámico, que permite construir consultas SQL en tiempo de ejecución (y ejecutarlas).

6.1.1 SQL incorporado

Conceptualmente, incluir comandos SQL en un programa anfitrión es sencillo. Las sentencias SQL (no las declaraciones) se pueden utilizar en cualquier lugar donde se permita una sentencia en el lenguaje anfitrión (con algunas restricciones). Las sentencias SQL deben estar claramente identificadas de manera que un preprocesador pueda tratarlas antes de invocar al compilador del lenguaje anfitrión. Además, cualquier variable del lenguaje anfitrión utilizada para pasar argumentos a un comando SQL debe declararse en SQL. En particular, algunas variables especiales del lenguaje anfitrión *deben* declararse en SQL (de forma que, por ejemplo, cualquier condición de error que se produzca durante la ejecución de SQL pueda comunicarse al programa de aplicación principal en el lenguaje anfitrión).

Sin embargo, hay dos complicaciones a tener en cuenta. En primer lugar, los tipos de datos reconocidos por SQL pueden no ser aceptados por el lenguaje anfitrión y viceversa. Este desajuste se soluciona habitualmente convirtiendo los valores de los datos antes de enviarlos a los comandos SQL (y viceversa). (SQL, como otros lenguajes de programación, proporciona un operador para convertir valores de un tipo en otro.) La segunda complicación tiene que ver con que SQL está **orientado a conjuntos**, y se resuelve mediante el uso de cursos (véase el Apartado 6.1.2).

En este apartado se asumirá por concreción que el lenguaje anfitrión es C, porque no hay grandes diferencias en cómo se incluyen sentencias SQL en diferentes lenguajes anfitriones.

Declaración de variables y excepciones

Las sentencias SQL se pueden referir a variables definidas en el programa anfitrión. Estas variables deben estar precedidas por un símbolo de dos puntos (:) en las sentencias SQL, y deben declararse entre los comandos EXEC SQL BEGIN DECLARE SECTION y EXEC SQL END DECLARE SECTION. Las declaraciones son similares a como aparecerían en un programa C y, como es habitual en C, se separan mediante punto y coma. Por ejemplo, se pueden declarar las variables *c_nombrem*, *c_idm*, *c_categoria*, y *c_edad* (la *c* inicial se utiliza como convenio para enfatizar que estas variables son variables del lenguaje anfitrión) de la siguiente forma:

```
EXEC SQL BEGIN DECLARE SECTION
char c_nombrem[20];
long c_idm;
short c_categoria;
float c_edad;
EXEC SQL END DECLARE SECTION
```

La primera pregunta que surge es qué tipos de SQL corresponden a los tipos de C, pues solamente se ha declarado una colección de variables C cuyos valores serán leídos (y posiblemente modificados) en un entorno de ejecución SQL cuando se ejecute una sentencia SQL que se refiera a ellos. El estándar SQL-92 define dicha correspondencia entre los tipos del lenguaje anfitrión y los tipos de SQL para una serie de lenguajes anfitriones. En este ejemplo, *c_idm* es de tipo **INTEGER**, *c_categoria* es de tipo **SMALLINT** y *c_edad* es de tipo **REAL**.

También se necesita una manera de que SQL informe de lo que ha ido mal si surge una condición de error cuando se ejecuta una sentencia SQL. El estándar SQL-92 considera dos variables especiales para comunicar errores, **SQLCODE** y **SQLSTATE**. **SQLCODE** es la más antigua de las dos y está definida para devolver un valor negativo cuando se produce una condición de error, sin especificar a qué error corresponde dicho número negativo. **SQLSTATE**, introducida en el estándar SQL-92 por primera vez, asocia valores predefinidos con diversas condiciones de error comunes, de manera que se introduce cierta uniformidad en la forma de informar de los errores. Una de estas dos variables *debe* declararse. El tipo C apropiado para **SQLCODE** es **long** y el tipo C para **SQLSTATE** es **char[6]**, una cadena de cinco caracteres. (Hay que tener en cuenta el terminador nulo de las cadenas de caracteres en C.) En este capítulo se asumirá que se declara **SQLSTATE**.

Incorporación de sentencias SQL

Todas las sentencias SQL incorporadas en un programa anfitrión deben identificarse claramente, dependiendo los detalles del lenguaje anfitrión; en C, las sentencias SQL deben estar precedidas por **EXEC SQL**. Una sentencia SQL puede aparecer fundamentalmente en cualquier lugar del programa anfitrión donde pueda aparecer una sentencia del lenguaje anfitrión.

Como ejemplo sencillo, la siguiente sentencia SQL incorporada introduce una fila en la relación Marineros en la que los valores de las columnas están basados en los valores de las variables del lenguaje anfitrión contenidas en la sentencia SQL:

```
EXEC SQL
  INSERT INTO Marineros VALUES (:c_nombrem, :c_idm, :c_categoria, :c_edad);
```

Se puede observar que el comando termina en un punto y coma, del mismo modo que se finalizan las sentencias en C.

La variable **SQLSTATE** debe comprobarse para detectar errores y excepciones después de cada sentencia SQL incorporada. SQL proporciona el comando **WHENEVER** para simplificar esta tarea tediosa:

```
EXEC SQL WHENEVER [ SQLERROR | NOT FOUND ] [ CONTINUE | GOTO sentencia ]
```

El propósito de esto es que el valor de **SQLSTATE** se compruebe después de la ejecución de cada sentencia SQL incorporada. Si se especifica **SQLERROR** y el valor de **SQLSTATE** indica una excepción, se transfiere el control a *sentencia*, que es la responsable del manejo de errores y excepciones. El control también se transfiere a *sentencia* si se especifica **NOT FOUND** y el valor de **SQLSTATE** es 02000, que indica **NO DATA** (no hay datos).

6.1.2 Cursos

Un problema importante en la incorporación de sentencias SQL en un lenguaje anfitrión como C es que se produce un *desajuste de impedancia* porque SQL opera sobre *conjuntos* de registros, mientras que lenguajes como C no soportan de forma limpia la abstracción “conjunto de registros”. La solución a esto consiste fundamentalmente en proporcionar un mecanismo que permita recuperar las filas de una relación de una en una.

Este mecanismo se denomina **cursor**. Se puede declarar un cursor sobre cualquier relación o cualquier consulta SQL (porque cualquier consulta devuelve un conjunto de filas). Una vez que se declara un cursor, se puede **abrir** (posicionando el cursor justo antes de la primera fila); **leer** la siguiente fila; **mover** el cursor (a la siguiente fila, a la fila después de las siguientes *n* filas, o a la fila anterior, etc., especificando parámetros adicionales al comando **FETCH**); o **cerrar** el cursor. Por tanto, un cursor permite fundamentalmente recuperar las filas de una tabla posicionando el cursor en una fila particular y leyendo su contenido.

Definición y uso básico de cursos

Los cursos permiten examinar desde el programa en el lenguaje anfitrión una colección de filas generadas por una sentencia SQL incorporada:

- Normalmente es necesario abrir un cursor si la sentencia incorporada es una consulta **SELECT** (es decir, una consulta). No obstante, se puede evitar abrir un cursor si la consulta contiene una sola fila, como se verá en breve.
- Las sentencias **INSERT**, **DELETE** y **UPDATE** normalmente no requieren un cursor, aunque algunas variantes de **DELETE** y **UPDATE** lo utilizan.

Como ejemplo se puede buscar el nombre y la edad de un marinero especificado mediante la asignación de un valor a la variable del lenguaje anfitrión *c_idm* declarada anteriormente, como se muestra a continuación:

```
EXEC SQL SELECT M.nombrem, M.edad
    INTO   :c_nombrem, :c_edad
    FROM   Marineros M
    WHERE  M.idm = :c_idm;
```

La cláusula **INTO** permite asignar las columnas de una fila resultado a las variables del lenguaje anfitrión *c_nombrem* y *c_edad*. Por tanto, no se necesita un cursor para incluir esta consulta en un programa anfitrión. Pero, ¿qué ocurre con la siguiente consulta, que obtiene los nombres y edades de todos los marineros con una categoría mayor que el valor actual de la variable del lenguaje anfitrión *c_categoríamín*?

```
SELECT M.nombrem, M.edad
FROM   Marineros M
WHERE  M.categoría > :c_categoríamín
```

Esta consulta devuelve una colección de filas, en lugar de una sola fila. Cuando se ejecuta interactivamente, las respuestas se muestran en la pantalla. Si se incluye esta consulta en un

programa C precediendo el comando con `EXEC SQL`, ¿cómo pueden relacionarse los resultados a las variables del lenguaje anfitrión? La cláusula `INTO` no es adecuada porque hay que gestionar varias filas. La solución está en el uso de un cursor:

```
DECLARE infom CURSOR FOR
SELECT M.nombrem, M.edad
FROM Marineros M
WHERE M.categoría > :c_categoríamín;
```

Este código puede incluirse en un programa C que al ejecutarse definirá el cursor `infom`. Posteriormente se puede abrir el cursor:

```
OPEN infom;
```

El valor de `c_categoríamín` en la consulta SQL asociada con el cursor es el valor de esta variable cuando se abre el cursor. (La declaración del cursor se procesa durante la compilación, mientras que el comando `OPEN` se ejecuta en tiempo de ejecución.)

Un cursor puede verse como “apuntando” a una fila en la colección de respuestas de la consulta asociada. Cuando se abre un cursor, éste se posiciona justo antes de la primera fila. Se puede utilizar el comando `FETCH` para leer la primera fila del cursor `infom` en las variables del lenguaje anfitrión:

```
FETCH infom INTO :c_nombrem, :c_edad;
```

Cuando se ejecuta la sentencia `FETCH`, el cursor se posiciona apuntando a la siguiente fila (que es la primera fila de la tabla cuando `FETCH` se ejecuta por primera vez después de abrir el cursor) y los valores de las columnas en la fila se copian en las variables correspondientes del lenguaje anfitrión. Ejecutando repetidamente la sentencia `FETCH` (por ejemplo, en un bucle `while` en el programa C), se pueden leer todas las filas de la consulta, de una en una. Existen parámetros adicionales del comando `FETCH` que permiten posicionar el cursor de manera muy flexible, aunque aquí no se estudiarán.

¿Cómo se puede saber cuándo se han recorrido todas las filas asociadas a un cursor? Examinando el contenido de las variables especiales `SQLCODE` o `SQLSTATE`, por supuesto. Por ejemplo, `SQLSTATE` toma el valor `02000` que denota `NO DATA`; esto indica que no hay más filas si la sentencia `FETCH` posiciona el cursor después de la última.

Cuando se ha terminado de utilizar un cursor, éste se puede cerrar:

```
CLOSE infom;
```

El cursor se puede abrir de nuevo si es necesario, y el valor de `:c_categoríamín` en la consulta SQL asociada al cursor será el valor de la variable del lenguaje anfitrión `c_categoríamín` en ese momento.

Propiedades de los cursores

La forma general de una declaración de cursor es:

```
DECLARE nombrecursor [INSENSITIVE] [SCROLL] CURSOR
[ WITH HOLD ]
```

```
FOR una consulta
[ ORDER BY lista-elem-orden ]
[ FOR READ ONLY | FOR UPDATE ]
```

Un cursor puede declararse como de **sólo lectura** (FOR READ ONLY) o **actualizable** (FOR UPDATE), si es un cursor sobre una relación base o una vista actualizable. Si es modificable, unas variantes simples de los comandos UPDATE y DELETE permiten actualizar o eliminar la fila sobre la que el cursor está posicionado. Por ejemplo, si *infom* es un cursor actualizable abierto, se puede ejecutar la siguiente sentencia:

```
UPDATE Marineros M
SET    M.categoría = M.categoría - 1
WHERE  CURRENT of infom;
```

Esta sentencia SQL incorporada modifica el valor de *categoría* de la fila en la que está actualmente posicionado el cursor *infom*; del mismo modo, se puede eliminar esta fila ejecutando la siguiente sentencia:

```
DELETE Marineros M
WHERE CURRENT of infom;
```

Un cursor es actualizable de manera predeterminada, excepto cuando es desplazable o insensible (véase más abajo), en cuyo caso es de sólo lectura.

Si se especifica la palabra clave SCROLL el cursor es **desplazable**, lo que significa que se pueden utilizar variaciones del comando FETCH para posicionar el cursor de forma muy flexible; de otro modo, sólo se permite el comando FETCH básico, que obtiene la siguiente fila del cursor.

Si se especifica la palabra clave INSENSITIVE, el cursor se comporta como si estuviera asociado a una copia privada de la colección de filas resultado. Si no se especifica, otras acciones de alguna transacción podrían modificar estas filas, produciendo un comportamiento impredecible. Por ejemplo, mientras se están leyendo filas mediante el cursor *infom* se podrían modificar valores de la columna *categoría* de las filas de Marineros ejecutando concurrentemente el comando:

```
UPDATE Marineros M
SET    M.categoría = M.categoría - 1
```

Considérese una fila de Marineros tal que (1) todavía no se ha leído, y (2) su valor original de *categoría* satisface la condición de la cláusula WHERE de la consulta asociada a *infom*, pero el nuevo valor de *categoría* no. ¿Se leerá esta fila de Marineros? Si se especifica INSENSITIVE, el comportamiento de la consulta es como si todas las respuestas fueran obtenidas cuando se abrió *infom*; por tanto, el comando de actualización no tiene efecto en las filas leídas por *infom* si se ejecuta después de abrir *infom*. Si no se especifica INSENSITIVE, el funcionamiento en esta situación depende de la implementación.

Se puede especificar que un cursor es **mantenible** con la cláusula WITH HOLD, que hace que el cursor no se cierre cuando se comprometa la transacción. Esta cláusula está motivada por transacciones largas que acceden (y posiblemente modifican) un gran número de filas de una tabla. Si se aborta la transacción por cualquier motivo, el sistema potencialmente tiene que

rehacer gran cantidad de trabajo cuando se reinicia la transacción. Incluso si la transacción no se aborta, sus bloqueos se mantienen durante mucho tiempo, y se reduce la concurrencia del sistema. La alternativa es dividir la transacción en transacciones más pequeñas, pero recordar la posición en la tabla entre transacciones (y otros detalles similares) es complejo y propenso a errores. Permitir que el programa comprometa la transacción iniciada, pero reteniendo el controlador de la tabla activa (es decir, el cursor) soluciona este problema: la aplicación puede comprometer la transacción e iniciar una nueva transacción, y de este modo guardar las modificaciones que ha hecho hasta ese momento.

Finalmente, ¿en qué orden obtienen las filas los comandos `FETCH`? En general, no está especificado este orden, pero puede utilizarse la cláusula opcional `ORDER BY` para especificar algún tipo de orden. Hay que tener en cuenta que las columnas que aparecen en la cláusula `ORDER BY` no pueden modificarse mediante el cursor.

La lista **lista-elem-orden** contiene elementos **elem-orden**; un elem-orden es un nombre de columna, seguido opcionalmente de una de las palabras clave `ASC` o `DESC`. Cada columna mencionada en la cláusula `ORDER BY` debe aparecer también en la lista **lista-de-selección** de la consulta asociada al cursor; de otro modo, no está claro sobre qué columnas debe ordenarse. Las palabras clave `ASC` o `DESC` que aparecen a continuación de un nombre de columna controlan si el resultado deber ordenarse —respecto a esa columna— en orden ascendente o descendente; por defecto es `ASC`. Esta cláusula se aplica como el último paso en la evaluación de la consulta.

Considérese la consulta del Apartado 5.5.1 y el resultado mostrado en la Figura 5.13. Supóngase que se ha abierto un cursor sobre esta consulta con la cláusula:

`ORDER BY edadmín ASC, categoría DESC`

El resultado se ordena primero en orden ascendente por `edadmín`, y si existen varias filas con el mismo valor para `edadmín`, éstas se ordenan en orden descendente según `categoría`. El cursor leerá las filas en el orden mostrado en la Figura 6.1.

<i>categoría</i>	<i>edadmín</i>
8	25,5
3	25,5
7	35,0

Figura 6.1 Orden en el que se leen las tuplas

6.1.3 SQL dinámico

Considérese una aplicación tal como una hoja de cálculo o una interfaz gráfica que necesita acceder a datos de un SGBD. Dicha aplicación debe aceptar comandos de un usuario y, en función de las necesidades del usuario, generar las sentencias SQL apropiadas para obtener los datos necesarios. En estas situaciones, puede no ser posible predecir por adelantado las sentencias SQL que se necesitarán ejecutar, incluso si hay (presumiblemente) algún algoritmo por el que la aplicación pueda construir las sentencias SQL necesarias una vez que se recibe un comando del usuario.

SQL proporciona algunas funcionalidades para estas situaciones: **SQL dinámico**. Se ilustran los dos comandos principales, **PREPARE** y **EXECUTE**, mediante un ejemplo sencillo:

```
char c_cadenasql[] = {"DELETE FROM Marineros WHERE categoría>5"};
EXEC SQL PREPARE listosParaSalir FROM :c_cadenasql;
EXEC SQL EXECUTE listosParaSalir;
```

La primera sentencia declara la variable C *c_cadenasql* e inicializa su valor a una cadena de caracteres con un comando SQL. La segunda sentencia analiza y compila esta cadena como un comando SQL, asociando el código ejecutable a la variable SQL *listosParaSalir*. (Como *listosParaSalir* es una variable SQL, como un nombre de cursor, no está precedida por un símbolo de dos puntos.) La tercera sentencia ejecuta el comando.

Existen muchas situaciones que requieren el uso de SQL dinámico. Sin embargo, obsérvese que la preparación de un comando de SQL dinámico ocurre en tiempo de ejecución y se produce una sobrecarga en la ejecución. Los comandos SQL interactivos e incorporados pueden prepararse una sola vez durante la compilación, y después ejecutarse tantas veces como se quiera. Por tanto, debería limitarse el uso de SQL dinámico a situaciones en las que resulte esencial.

Hay muchas más cosas que conocer sobre SQL dinámico —el paso de parámetros del programa anfitrión a la consulta SQL que se está preparando, por ejemplo— pero no se profundizará en ellos.

6.2 UNA INTRODUCCIÓN A JDBC

SQL incorporado permite la integración de SQL con un lenguaje de programación de propósito general. Como se describe en el Apartado 6.1.1, un preprocesador específico del SGBD transforma las sentencias de SQL incorporado en llamadas a funciones del lenguaje anfitrión. Los detalles de esta traducción varían de un SGBD a otro y, por tanto, aunque el código fuente pueda compilarse para trabajar con diferentes SGBD, el ejecutable final sólo funciona con un SGBD determinado.

ODBC y **JDBC**, siglas de Open DataBase Connectivity y Java DataBase Connectivity, también permiten la integración de SQL con un lenguaje de programación de propósito general. Tanto ODBC como JDBC permiten al programador de aplicaciones acceder a las características de la base de datos de una forma estandarizada mediante una **interfaz de programación de aplicaciones** (API, application programming interface). A diferencia de SQL incorporado, ODBC y JDBC permiten el uso de un único ejecutable para acceder a diferentes SGBD *sin recompilación*. Por tanto, mientras que SQL incorporado es independiente del SGBD solamente respecto al código fuente, las aplicaciones que usan ODBC o JDBC son independientes del SGBD tanto respecto al código fuente como respecto al ejecutable. Además, utilizando ODBC o JDBC, una aplicación puede utilizar varios SGBD diferentes simultáneamente.

ODBC y JDBC consiguen portabilidad respecto al ejecutable introduciendo un nivel extra de indirección. Cualquier interacción directa con un SGBD determinado ocurre a través de un **controlador** específico del SGBD. Un controlador es un programa que traduce las llamadas ODBC o JDBC en llamadas específicas del SGBD. Los controladores se cargan dinámicamente

Controladores JDBC. La fuente más actualizada de controladores JDBC es la página de controladores JDBC de Sun

<http://industry.java.sun.com/products/jdbc/drivers>

En este sitio se encuentran disponibles los controladores JDBC para los sistemas de base de datos más importantes.

bajo demanda, ya que los SGBD que la aplicación va a utilizar sólo se conocen en tiempo de ejecución. Los controladores disponibles se registran con un **gestor de controladores**.

Un aspecto interesante a señalar es que un controlador no tiene que interactuar necesariamente con un SGBD que entienda SQL. Es suficiente que el controlador traduzca los comandos SQL de la aplicación en los comandos equivalentes que el SGBD entiende. Por tanto, en lo que queda de este apartado, nos referiremos al subsistema de almacenamiento de datos con el que un controlador interactúa como una **fuente de datos**.

Una aplicación que interactúa con una fuente de datos a través de ODBC o JDBC selecciona la fuente de datos, carga dinámicamente el controlador correspondiente y establece una conexión con la fuente de datos. No hay límite sobre el número de conexiones abiertas, y una aplicación puede tener varias conexiones abiertas con fuentes de datos diferentes. Cada conexión tiene semántica de transacción; es decir, los cambios de una conexión sólo son visibles para otras conexiones después de que la primera ha comprometido sus cambios. Mientras que una conexión está abierta, las transacciones se ejecutan enviando sentencias SQL, obteniendo resultados, procesando errores y finalmente comprometiendo la transacción o cancelándola. La aplicación se desconecta de la fuente de datos para terminar la interacción.

El resto de este capítulo se centra en JDBC.

6.2.1 Arquitectura

La arquitectura de JDBC tiene cuatro componentes principales: la *aplicación*, el *gestor de controladores*, varios *controladores* específicos de fuentes de datos y las correspondientes *fuentes de datos*.

La *aplicación* inicia y finaliza la conexión con una fuente de datos. Ella asigna los límites de las transacciones, envía sentencias SQL y recupera los resultados —todo ello por medio de una interfaz bien definida, según está especificado en el API de JDBC—. El objetivo principal del *gestor de controladores* es cargar controladores JDBC y pasar llamadas a funciones JDBC de la aplicación al controlador correcto. El gestor de controladores también gestiona las llamadas de inicialización y de información de las aplicaciones y puede registrar las llamadas a funciones. Además, el gestor de controladores realiza algunas comprobaciones de errores rudimentarias. El *controlador* establece la conexión con la fuente de datos. Además de enviar peticiones y devolver resultados de peticiones, el controlador traduce los datos, los formatos de error y los códigos de error de una forma específica de la fuente de datos en el estándar JDBC. La *fuente de datos* procesa los comandos del controlador y devuelve los resultados.

Dependiendo de la ubicación relativa de la fuente de datos y la aplicación, son posibles varios escenarios. Los controladores se clasifican según JDBC en cuatro tipos en función de la relación arquitectónica entre la aplicación y la fuente de datos:

- **Tipo I — Puentes.** Este tipo de controlador traduce las llamadas a funciones JDBC en llamadas a funciones de otro API que no es nativo del SGBD. Un ejemplo es un puente JDBC-ODBC; una aplicación puede utilizar llamadas JDBC para acceder a una fuente de datos compatible ODBC. La aplicación carga solamente un controlador, el puente. Los puentes tienen la ventaja de que es fácil hacer funcionar la aplicación sobre una instalación existente, y no es necesario instalar nuevos controladores. Pero el uso de puentes tiene varios inconvenientes. El mayor número de capas entre la fuente de datos y la aplicación afecta al rendimiento. Además, el usuario está limitado a la funcionalidad que permite el controlador ODBC.
- **Tipo II — Traducción directa al API nativo a través de un controlador no hecho en Java.** Este tipo de controlador traduce las llamadas a funciones JDBC directamente a invocaciones de métodos del API de una fuente de datos específica. El controlador normalmente está escrito utilizando una combinación de C++ y Java; se enlaza dinámicamente y para la fuente de datos específica. Esta arquitectura es significativamente más eficiente que un puente JDBC-ODBC. Pero tiene la desventaja de que el controlador de la base de datos que implementa el API debe estar instalado en cada ordenador que ejecute la aplicación.
- **Tipo III — Puentes de red.** El controlador se comunica a través de una red a un servidor intermedio que traduce las peticiones JDBC en invocaciones a métodos específicos del SGBD. En este caso, el controlador de la parte cliente (es decir, el puente de red) no es específico del SGBD. Este controlador puede ser muy pequeño, pues la única funcionalidad que necesita implementar consiste en el envío de sentencias SQL al servidor intermedio. Este servidor puede utilizar un controlador JDBC de tipo II para conectar con la fuente de datos.
- **Tipo IV — Traducción directa al API nativo a través de un controlador Java.** En lugar de llamar directamente al API del SGBD, el controlador se comunica con el SGBD mediante sockets Java. En este caso, el controlador del lado del cliente está escrito en Java, pero es específico del SGBD. Traduce las llamadas JDBC al API nativo del sistema de base de datos. Esta solución no requiere ninguna capa intermedia, y como toda la implementación es Java, su rendimiento es normalmente bastante bueno.

6.3 CLASES E INTERFACES JDBC

JDBC es una colección de clases e interfaces Java que permiten el acceso a bases de datos desde programas escritos en Java. Contiene métodos para conectarse a una fuente de datos remota, ejecutar sentencias SQL, examinar conjuntos de resultados de sentencias SQL, gestionar transacciones y manejar excepciones. Las clases e interfaces forman parte del paquete `java.sql`. Por tanto, todos los fragmentos de código de este apartado deben incluir la sentencia `import java.sql.*` al principio del código; se omitirá esta sentencia en lo que resta de apartado. JDBC 2.0 también incluye el paquete `javax.sql`, el **paquete opcional JDBC**. Este paquete añade, entre otras cosas, la posibilidad de utilizar colas de conexiones y la interfaz `RowSet`. Las colas de conexiones se estudiarán en el Apartado 6.3.2 y la interfaz `ResultSet` en el Apartado 6.3.4.

Ahora se ilustrarán los pasos necesarios para enviar una consulta de base de datos a una fuente de datos y obtener los resultados.

6.3.1 Gestión de controladores JDBC

Los controladores de fuentes de datos se gestionan en JDBC con la clase `DriverManager`, que mantiene una lista con todos los controladores cargados actualmente. Esta clase dispone de los métodos `registerDriver`, `deregisterDriver` y `getDrivers` para poder agregar y eliminar controladores.

El primer paso para conectar con una fuente de datos es cargar el controlador JDBC correspondiente. Esto se realiza utilizando el mecanismo de Java para cargar clases dinámicamente. El método estático `forName` de la clase `Class` devuelve la clase Java que se especifica en la cadena de caracteres pasada como argumento, y ejecuta su constructor `static`. El constructor estático de la clase que se ha cargado dinámicamente carga a su vez un ejemplar de la clase `Driver`, y este objeto `Driver` se registra a sí mismo mediante la clase `DriverManager`.

El siguiente código de ejemplo Java carga explícitamente un controlador JDBC:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Hay dos formas más de registrar un controlador. Se puede incluir el controlador con la opción `-Djdbc.drivers=oracle/jdbc.driver` en la línea de comandos cuando se inicia la aplicación Java. O bien se puede instanciar un controlador explícitamente, aunque este método no se utiliza habitualmente, pues el nombre del controlador debe especificarse en el código de la aplicación, y por ello la aplicación deja de ser independiente del controlador.

Después de registrar el controlador, podemos conectar con la fuente de datos.

6.3.2 Conexiones

Una sesión con una fuente de datos se inicia con la creación de un objeto `Connection`. Una conexión identifica una sesión lógica con una fuente de datos; diferentes conexiones dentro del mismo programa Java se pueden referir a la misma o a diferentes fuentes de datos. Las conexiones se especifican por medio de un **URL JDBC**, un URL que usa el protocolo JDBC. Este URL tiene la forma:

```
jdbc:<subprotocolo>:<otrosParámetros>
```

El ejemplo de código de la Figura 6.2 establece una conexión con una base de datos Oracle asumiendo que las cadenas de caracteres `idUsuario` y `clave` están asignadas a valores válidos.

Las conexiones en JDBC pueden tener diferentes propiedades. Por ejemplo, una conexión puede especificar la granularidad de las transacciones. Si se especifica `autocommit` en una conexión, cada sentencia SQL se considera como incluida en su propia transacción. Si `autocommit` está deshabilitado, entonces la serie de sentencias que formen una transacción se pueden comprometer con el método `commit()` de la clase `Connection`, o abortarse utilizando el método `rollback()`. La clase `Connection` tiene métodos para asignar el modo de autocompromiso (`Connection.setAutoCommit`) y para obtener el modo de autocompromiso actual (`getAutoCommit`). Los siguientes métodos son parte de la interfaz `Connection` y permiten asignar y obtener otras propiedades:

```

String url = "jdbc:oracle:www.tiendalibros.com:3083"
Connection conexion;
try {
    Connection conexion =
        DriverManager.getConnection(url,idUsuario,clave);
}
catch(SQLException excpt) {
    System.out.println(excpt.getMessage());
    return;
}

```

Figura 6.2 Establecimiento de una conexión con JDBC

Conexiones JDBC. Hay que recordar cerrar las conexiones a fuentes de datos y devolver las conexiones compartidas a la cola de conexiones. Los sistemas de base de datos tienen un número limitado de recursos disponibles para las conexiones, y a menudo las conexiones huérfanas sólo pueden detectarse mediante tiempos de espera —y mientras el sistema de base de datos está esperando a que venza el tiempo de espera de la conexión se desperdician los recursos utilizados por la conexión huérfana—.

- `public int getTransactionIsolation() throws SQLException` y
`public void setTransactionIsolation(int l) throws SQLException`.
Estas funciones obtienen y asignan el nivel actual de aislamiento para las transacciones manejadas en la conexión actual. Se pueden utilizar los cinco niveles de aislamiento SQL (véase el Apartado 8.6 para una descripción completa) asignando el parámetro *l* a uno de los siguientes valores:
 - `TRANSACTION_NONE`
 - `TRANSACTION_READ_UNCOMMITTED`
 - `TRANSACTION_READ_COMMITTED`
 - `TRANSACTION_REPEATABLE_READ`
 - `TRANSACTION_SERIALIZABLE`
- `public boolean getReadOnly() throws SQLException` y
`public void setReadOnly(boolean readOnly) throws SQLException`.
Estas funciones permiten al usuario especificar si las transacciones a ejecutar en esta conexión son de sólo lectura.
- `public boolean isClosed() throws SQLException`.
Comprueba si la conexión actual ha sido cerrada.
- `setAutoCommit` y `get AutoCommit`.
Estas dos funciones han sido comentadas anteriormente.

Establecer una conexión con una fuente de datos es una operación costosa, ya que implica varios pasos, tales como el establecimiento de una conexión de red con la fuente de datos, autenticación y asignación de recursos, como por ejemplo memoria. En el caso de que una aplicación establezca muchas conexiones diferentes de diferentes partes (como puede ser un servidor Web), las conexiones a menudo se reúnen en una cola para evitar esta sobrecarga. Una **cola de conexiones** es un conjunto de conexiones establecidas con una fuente de datos. Siempre que se necesite una nueva conexión, se utiliza una de las conexiones de la cola, en lugar de crear una nueva conexión con la fuente de datos.

Las colas de conexiones se pueden tratar tanto con código especializado en la aplicación, como con el paquete opcional `javax.sql`, que proporciona funcionalidad para manejar colas de conexiones y permite asignar diversos parámetros, como la capacidad de la memoria y los factores de contracción y expansión. La mayor parte de los servidores de aplicaciones (véase el Apartado 7.7.2) implementan el paquete `javax.sql` o bien una variante propietaria.

6.3.3 Ejecución de sentencias SQL

Ahora se explicará la creación y ejecución de sentencias SQL con JDBC. En los ejemplos de código de JDBC de este apartado asumiremos un objeto `Connection` denominado `con`. JDBC soporta tres formas diferentes de ejecutar sentencias: `Statement`, `PreparedStatement` y `CallableStatement`. La clase `Statement` es la clase base para las otras dos clases. Permite consultar la fuente de datos con cualquier consulta SQL estática o dinámicamente generada. Aquí se verá la clase `PreparedStatement` aquí, y la clase `CallableStatement` en el Apartado 6.5, cuando se examinen los procedimientos almacenados.

La clase `PreparedStatement` genera dinámicamente sentencias SQL precompiladas que se pueden utilizar varias veces; estas sentencias SQL pueden contener parámetros, pero su estructura queda fijada cuando se crea el objeto `PreparedStatement` (que representa la sentencia SQL).

Considérese el código de ejemplo de la Figura 6.3 que usa un objeto `PreparedStatement`. La consulta SQL especifica la cadena de caracteres con la consulta, pero utiliza “?” para los valores de los parámetros, que se asignan posteriormente utilizando los métodos `setString`, `setFloat` y `setInt`. Los marcadores de sustitución “?” se pueden utilizar en cualquier lugar de una sentencia SQL donde pueden reemplazarse por un valor. Ejemplos de lugares en los que pueden aparecer incluyen la cláusula `WHERE` (por ejemplo, “`WHERE autor=?`”), o en las sentencias SQL `UPDATE` y `INSERT`, como en la Figura 6.3. El método `setString` es una forma de asignar el valor de un parámetro; existen métodos análogos para `int`, `float` y `date`. Es una buena norma de estilo utilizar siempre `clearParameters()` antes de asignar valores de parámetros, con el objetivo de eliminar cualquier dato anterior.

Existen diferentes formas de enviar a ejecutar la consulta a la fuente de datos. En el ejemplo anterior, se ha usado el comando `executeUpdate`, que se utiliza si se conoce que la sentencia SQL no devuelve ningún registro (las sentencias SQL `UPDATE`, `INSERT`, `ALTER` y `DELETE`). El método `executeUpdate` devuelve un número entero que indica el número de filas que se han modificado con la sentencia SQL; devuelve 0 cuando se ejecuta con éxito sin modificar ninguna fila.

```

// la cantidad inicial siempre es 0
String sql = "INSERT INTO Libros VALUES(?, ?, ?, ?, 0, ?)";
PreparedStatement pstmt = con.prepareStatement(sql);

// ahora se instancian los parámetros con valores
// se asume que isbn, título, etc. son variables Java que
// contienen los valores que serán introducidos
pst.clearParameters();
pst.setString(1, isbn);
pst.setString(2, título);
pst.setString(3, autor);
pst.setFloat(5, precio);
pst.setInt(6, año);

int numRows = pstmt.executeUpdate();

```

Figura 6.3 Actualización SQL con un objeto `PreparedStatement`

El método `executeQuery` se utiliza si la sentencia SQL devuelve datos, como por ejemplo en consultas `SELECT`. JDBC tiene su propio mecanismo de cursos por medio del objeto `ResultSet`, que se estudiará a continuación. El método `execute` es más general que `executeQuery` y `executeUpdate`; las referencias al final del capítulo proporcionan punteros con más detalles.

6.3.4 ResultSets

Como se ha comentado en el apartado anterior, la sentencia `executeQuery` devuelve un objeto de tipo `ResultSet`, que es similar a un cursor. Los cursos `ResultSet` en JDBC 2.0 son muy potentes; permiten recorrer los registros hacia adelante y hacia atrás, así como editar e insertar filas en el cursor.

En su forma más básica, el objeto `ResultSet` permite leer una fila del resultado de la consulta cada vez. Inicialmente, el `ResultSet` está posicionado antes de la primera fila, y hay que recuperar la primera fila con una llamada explícita al método `next()`. Este método devuelve `false` si no hay más filas en la respuesta de la consulta, y `true` en otro caso. El fragmento de código mostrado en la Figura 6.4 ilustra el uso básico de un objeto `ResultSet`.

Mientras que `next()` permite obtener la fila siguiente en el resultado de la consulta, también es posible moverse por el resultado de la consulta de otras formas:

- `previous()` mueve a la fila anterior.
- `absolute(int num)` mueve a la fila con el número especificado.
- `relative(int num)` mueve hacia delante o hacia atrás (si `num` es negativo) relativo a la posición actual. `relative(-1)` tiene el mismo efecto que `previous()`.

```

String sqlQuery;
ResultSet rs=st.executeQuery(consultaSql);
// rs ahora es un cursor
// la primera llamada a rs.next() mueve al primer registro
// rs.next() mueve al siguiente registro
while (rs.next()) {
    // proceso del registro
}

```

Figura 6.4 Uso de un objeto `ResultSet`

- `first()` mueve a la primera fila, y `last()` a la última.

Correspondencia de tipos de datos Java y SQL

Al considerar la interacción de una aplicación con una fuente de datos, surgen los mismos problemas que ya encontramos en SQL incorporado (por ejemplo, cómo pasar información entre la aplicación y la fuente de datos a través de variables compartidas). Para tratar estos problemas, JDBC proporciona tipos de datos especiales y especifica su relación con los tipos de datos SQL correspondientes. La Figura 6.5 muestra los **métodos de acceso** de un objeto `ResultSet` para los tipos datos SQL más comunes. Con estos métodos de acceso se pueden obtener los valores de la fila actual del resultado de la consulta referenciado por el objeto `ResultSet`. Cada método de acceso tiene dos variantes: una de ellas obtiene los valores por el índice de la columna, comenzando por uno, y la otra obtiene los valores por el nombre de la columna. El ejemplo siguiente muestra cómo acceder a los campos de la fila actual de un `ResultSet` utilizando estos métodos.

```

String consultaSql;
ResultSet rs = st.executeQuery(consultaSql);
while (rs.next()) {
    isbn = rs.getString(1);
    título = rs.getString("TÍTULO");
    // proceso de isbn y título
}

```

6.3.5 Excepciones y avisos

De forma similar a la variable `SQLSTATE`, la mayor parte de los métodos de `java.sql` pueden lanzar excepciones del tipo `SQLException` si ocurre un error. La información incluye `SQLState`, una cadena de caracteres que describe el error (por ejemplo, si la sentencia contiene un error de sintaxis de SQL). Además del método estándar `getMessage()` heredado de

Tipo SQL	Clase Java	Método get de ResultSet
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.Timestamp	getTimestamp()

Figura 6.5 Lectura de tipos de datos SQL de un objeto ResultSet

`Throwable`, `SQLException` tiene dos métodos adicionales que proporcionan más información, y un método para obtener (o encadenar) excepciones adicionales:

- `public String getSQLState()` devuelve un identificador SQLState basado en la especificación SQL:1999, como se ha comentado en el Apartado 6.1.1.
- `public int getErrorCode()` obtiene un código de error específico del SGBD.
- `public SQLException getNextException()` obtiene la siguiente excepción en una cadena de excepciones asociada al objeto `SQLException` actual.

`SQLWarning` es una subclase de `SQLException`. Los avisos (warnings) no son tan graves como los errores, y el programa normalmente puede continuar sin tratarlos de forma especial. Los avisos no se lanzan como las otras excepciones, y no son capturados como parte de un bloque `try--catch` dispuesto en la sentencia `java.sql`. Es necesario comprobar expresamente si existe un aviso. Los objetos `Connection`, `Statement` y `ResultSet` disponen del método `getWarnings()` con el que se pueden obtener avisos SQL, si hay alguno. Se puede evitar que se recuperen avisos duplicados utilizando `clearWarnings()`. Por su parte, los objetos `Statement` limpian los avisos automáticamente cuando se ejecuta la siguiente sentencia; los objetos `ResultSet` limpian los avisos cada vez que se accede a una nueva tupla.

El código habitual para obtener `SQLWarnings` es similar al mostrado en la Figura 6.6.

6.3.6 Obtención de los metadatos de una base de datos

Se puede utilizar el objeto `DatabaseMetaData` para obtener información sobre el sistema de base de datos, así como información del catálogo de la base de datos. Por ejemplo, el siguiente fragmento de código muestra el modo de obtener el nombre y la versión del controlador JDBC:

```
DatabaseMetaData md = con.getMetaData();

System.out.println("Información del controlador:");
System.out.println("Nombre:" + md.getDriverName()
+ "; versión:" + md.getDriverVersion());
```

```

try {
    st = con.createStatement();
    warning = con.getWarnings();
    while( warning != null) {
        // manejarAvisosSQL
        warning = warning.getNextWarning();           //código para procesar el aviso
    }                                                 //obtiene siguiente aviso
    con.clearWarnings();

    st.executeUpdate( queryString );
    warning = st.getWarnings();
    while( warning != null) {
        // manejarAvisosSQL
        warning = warning.getNextWarning();           //código para procesar el aviso
    }                                                 //obtiene siguiente aviso
}
} // fin de try
catch ( SQLException SQLe) {
    // código para manejar excepción
} // fin de catch

```

Figura 6.6 Procesamiento de avisos y excepciones con JDBC

El objeto `DatabaseMetaData` tiene muchos otros métodos (en JDBC 2.0, exactamente 134); algunos de ellos se listan a continuación:

- `public ResultSet getCatalogs() throws SQLException`. Esta función devuelve un `ResultSet` que se puede usar para recorrer los catálogos. Las funciones `getIndexInfo()` y `getTables()` se comportan de la misma forma.
- `public int getMaxConnections() throws SQLException`. Esta función determina el número máximo posible de conexiones.

Este apartado sobre JDBC concluye con un fragmento de código de ejemplo que examina todos los metadatos, mostrado en la Figura 6.7.

6.4 SQLJ

SQLJ (siglas de ‘SQL-Java’) fue desarrollado por el Grupo SQLJ, un grupo de fabricantes de bases de datos y Sun. SQLJ fue desarrollado para complementar la forma dinámica de crear consultas de JDBC con un modelo estático. Por tanto, es muy parecido a SQL incorporado. A diferencia de JDBC, disponer de consultas SQL semiestáticas permite al compilador realizar comprobaciones de la sintaxis SQL, comprobaciones rigurosas de compatibilidad de tipos de las variables del lenguaje anfitrión con los atributos SQL correspondientes, y la consistencia de la consulta con el esquema de base de datos —tablas, atributos, vistas y procedimientos

```

DatabaseMetaData dmd = con.getMetaData();
ResultSet rsTablas = dmd.getTables(null,null,null,null);
String nombreTabla;

while(rsTablas.next()) {
    nombreTabla = rsTablas.getString("TABLE_NAME");

    // Muestra los atributos de esta tabla
    System.out.println("Los atributos de la tabla "
        + nombreTabla + " son:");
    ResultSet rsColumnas = dmd.getColumns(null,null,nombreTabla, null);
    while (rsColumnas.next()) {
        System.out.print(rsColumnas.getString("COLUMN_NAME")
            + " ");
    }
}

// Muestra las claves primarias de esta tabla
System.out.println("Las claves de la tabla " + nombreTabla + " son:");
ResultSet rsClaves = dmd.getPrimaryKeys(null,null,nombreTabla);
while (rsClaves.next()) {
    System.out.print(rsClaves.getString("COLUMN_NAME") + " ");
}
}

```

Figura 6.7 Obtención de información sobre una fuente de datos

almacenados— todo durante la compilación. Por ejemplo, tanto en SQLJ como en SQL incorporado, las variables del lenguaje anfitrión siempre están ligadas estéticamente a los mismos argumentos, mientras que en JDBC es necesario utilizar sentencias separadas para ligar cada variable a un argumento y recuperar el resultado. Por ejemplo, la siguiente sentencia SQLJ liga las variables del lenguaje anfitrión *título*, *precio* y *autor* a los valores actuales del cursor *libros*.

```

#sql libros = {
    SELECT título, precio INTO :título, :precio
    FROM Libros WHERE autor = :autor
};

```

En JDBC se puede decidir dinámicamente las variables del lenguaje anfitrión que se almacenarán el resultado de la consulta. En el siguiente ejemplo se guarda el título del libro en la variable *ftítulo* si el libro fue escrito por Feynman, y en la variable *otítulo* en otro caso:

```
// se asume disponible un cursor ResultSet en rs
```

```

autor = rs.getString(3);

if (autor=="Feynman") {
    ftítulo = rs.getString(2);
}
else {
    otítulo = rs.getString(2);
}

```

Al escribir aplicaciones SQLJ basta con escribir código Java normal e incluir sentencias SQL de acuerdo a un conjunto de reglas. Las aplicaciones SQLJ se preprocesan con un programa traductor que reemplaza el código SQLJ con llamadas a una librería Java de SQLJ. El código modificado puede entonces compilarse con cualquier compilador Java. La librería de SQLJ normalmente hace llamadas a un controlador JDBC que realiza la conexión con el sistema de base de datos.

Existe una diferencia filosófica importante entre SQL incorporado, SQLJ y JDBC. Como los fabricantes de SGBD proporcionan sus propias versiones propietarias de SQL, es recomendable escribir las consultas SQL de acuerdo a los estándares SQL-92 o SQL:1999. SQLJ y JDBC obligan a ajustarse a los estándares, y el código resultante es mucho más portable entre diferentes sistemas de base de datos.

En lo que queda de este apartado se introducirá brevemente SQLJ.

6.4.1 Escritura de código SQLJ

SQLJ se introducirá mediante ejemplos, comenzando con un fragmento de código SQLJ que selecciona registros de la tabla Libros que corresponden a un autor determinado.

```

String título; Float precio; String autor;
#sql iterator Libros (String título, Float precio);
Libros libros;

// La aplicación asigna el autor,
// ejecuta la consulta y abre el cursor
#sql libros = {
    SELECT título, precio INTO :título, :precio
    FROM Libros WHERE autor = :autor
};

// recupera resultados
while (libros.next()) {
    System.out.println(libros.título() + "," + libros.precio());
}
libros.close();

```

El fragmento de código JDBC correspondiente es como sigue (asumiendo que también se han declarado `precio`, `nombre` y `autor`):

```

PreparedStatement st = connection.prepareStatement(
    "SELECT título, precio FROM Libros WHERE autor = ?");

// asigna el parametro en la consulta y la ejecuta
st.setString(1, autor);
ResultSet rs = st.executeQuery();

// recupera los resultados
while (rs.next()) {
    System.out.println(rs.getString(1) + ", " + rs.getFloat(2));
}

```

Comparando el código JDBC y SQLJ, se puede ver que el código SQLJ es mucho más fácil de leer que el código JDBC. Por tanto, SQLJ reduce el coste de desarrollo y mantenimiento de programas.

Consideremos los componentes individuales del código SQLJ con más detalle. Todas las sentencias SQLJ tienen el prefijo especial `#sql`. En SQLJ se recuperan los resultados de consultas SQL con objetos **iterator**, que básicamente son cursos. Un iterador es un ejemplar de una clase iterator. El uso de un iterador en SQLJ se realiza en cinco pasos:

- **Declarar la clase iterator.** En el código precedente esto se realiza con la sentencia `#sql iterator Libros (String título, Float precio);`
Esta sentencia crea una nueva clase Java que se puede utilizar para instanciar objetos.
- **Instanciar un objeto iterador de la nueva clase iterator.** El iterador se instancia en la sentencia `Libros libros;`.
- **Inicializar el iterador utilizando una sentencia SQL.** En el ejemplo, esto ocurre en la sentencia `#sql libros = ...`
- **Iterativamente, leer las filas del objeto iterador.** Este paso es muy similar a la lectura de filas mediante un objeto `ResultSet` en JDBC.
- **Cerrar el objeto iterador.**

Hay dos tipos de clase iterator: iteradores con nombre e iteradores posicionales. En los **iteradores con nombre**, se especifica tanto el tipo de variable como el nombre de cada columna del iterador. Esto permite recuperar columnas individualmente por nombre como en el ejemplo anterior en el que era posible recuperar la columna título de la tabla Libros utilizando la expresión `libros.título()`. En los **iteradores posicionales**, es necesario especificar solamente el tipo de variable de cada columna del iterador. Para acceder a las columnas del iterador se usa una construcción `FETCH ... INTO` similar a la de SQL incorporado. Ambos tipos de iteradores tienen la misma eficiencia; el iterador a utilizar depende del gusto del programador.

Volviendo al ejemplo es posible convertir el iterador en posicional mediante la siguiente sentencia:

```
#sql iterator Libros (String, Float);
```

A continuación se recuperan las filas del iterador de la siguiente forma:

```
while (true) {
    #sql { FETCH :libros INTO :título, :precio, };
    if (libros.endFetch()) {
        break;
    }
    // procesar el libro
}
```

6.5 PROCEDIMIENTOS ALMACENADOS

A menudo es importante ejecutar algunas partes de la lógica de la aplicación directamente en el espacio de proceso del sistema de base de datos. Ejecutar la lógica de la aplicación directamente en la base de datos tiene la ventaja de que la cantidad de datos que es transferida entre el servidor de la base de datos y el cliente que envió la sentencia SQL puede minimizarse, a la vez que se utiliza toda la potencia del servidor de la base de datos.

Cuando se ejecutan sentencias SQL desde una aplicación remota, los registros resultado de la consulta deben transferirse desde la base de datos a la aplicación. Si se utiliza un cursor para acceder remotamente a los resultados, el SGBD tiene recursos asignados tales como bloqueos y memoria mientras que se procesan los registros recuperados mediante el cursor. Por el contrario, un **procedimiento almacenado** es un programa que se llama con una única sentencia SQL y que puede ejecutarse localmente dentro del espacio de proceso del servidor de la base de datos. Los resultados pueden empaquetarse en un resultado más grande y devolverse a la aplicación, o la lógica de la aplicación puede realizarse directamente en el servidor, sin necesidad de transmitir los resultados al cliente.

Los procedimientos almacenados también son beneficiosos por razones de ingeniería del software. Una vez que se registra un procedimiento almacenado en el servidor de base de datos, diferentes usuarios pueden reutilizarlo, evitando duplicar esfuerzos escribiendo consultas SQL o la lógica de la aplicación, y haciendo de este modo el mantenimiento de código más fácil. Además, los programadores de aplicaciones no necesitan conocer el esquema de base de datos si se encapsulan todos los accesos a la base de datos en procedimientos almacenados.

Aunque se denominen *procedimientos* almacenados, no es necesario que sean procedimientos en el sentido de un lenguaje de programación; pueden ser funciones.

6.5.1 Creación de un procedimiento almacenado simple

En la Figura 6.8 se muestra un ejemplo de procedimiento almacenado escrito en SQL mostrado. Se puede observar que los procedimientos almacenados deben tener un nombre; este procedimiento tiene el nombre ‘MostrarNúmeroDePedidos’. Por lo demás, solamente contiene una sentencia SQL que se precompila y almacena en el servidor.

Los procedimientos almacenados también pueden tener parámetros. Éstos deben ser tipos SQL válidos, y deben tener uno de los siguientes **modos**: IN, OUT o INOUT. Los parámetros IN son argumentos para los procedimientos almacenados. Los parámetros OUT son resultados

```

CREATE PROCEDURE MostrarNumeroDePedidos
SELECT C.idc, C.nombrec, COUNT(*)
      FROM Clientes C, Pedidos P
     WHERE C.idc = P.idc
   GROUP BY C.idc, C.nombrec

```

Figura 6.8 Procedimiento almacenado en SQL

del procedimiento almacenado; éste asigna valores a todos los parámetros OUT que el usuario pueda procesar. Los parámetros INOUT combinan las propiedades de los parámetros IN y OUT: contienen valores a pasar al procedimiento almacenado, y éste puede modificarlos para devolverlos como valores de retorno. Los procedimientos almacenados imponen un ajuste de tipos estricto: si un parámetro es de tipo INTEGER, no puede ser llamado con un argumento de tipo VARCHAR.

A continuación se examinará un ejemplo de un procedimiento almacenado con argumentos. El procedimiento mostrado en la Figura 6.9 tiene dos argumentos: isbn.libro y cantAgregada. Actualiza el número de copias disponibles de un libro con la cantidad de un nuevo envío.

```

CREATE PROCEDURE AgregarInventario (
      IN isbn.libro CHAR(10),
      IN cantAgregada INTEGER)
UPDATE Libros
      SET stock = stock + cantAgregada
     WHERE isbn.libro = isbn

```

Figura 6.9 Procedimiento almacenado con argumentos

Los procedimientos almacenados no tienen por qué estar escritos en SQL, pueden escribirse en cualquier lenguaje anfitrión. Por ejemplo, el procedimiento mostrado en la Figura 6.10 es una función Java que el servidor de base de datos ejecuta dinámicamente siempre que la llama el cliente:

```

CREATE PROCEDURE ClasificarClientes(IN numero INTEGER)
LANGUAGE Java
EXTERNAL NAME 'file:///c:/procAlmacenados/clasif.jar'

```

Figura 6.10 Procedimiento almacenado en Java

6.5.2 Llamada a procedimientos almacenados

Los procedimientos almacenados pueden llamarse en SQL interactivo con la sentencia CALL:

```
CALL nombreProcAlmacenado(argumento1, argumento2, ..., argumentoN);
```

En SQL incorporado, los argumentos de un procedimiento almacenado normalmente son variables del lenguaje anfitrión. Por ejemplo, el procedimiento almacenado `agregarInventario` se llamaría de la siguiente forma:

```
EXEC SQL BEGIN DECLARE SECTION
char isbn[10];
long cantidad;
EXEC SQL END DECLARE SECTION

// asigna valores a isbn y cantidad
EXEC SQL CALL agregarInventario(:isbn,:cantidad);
```

Llamada a procedimientos almacenados desde JDBC

Con la clase `CallableStatement` es posible llamar a procedimientos almacenados desde JDBC. Esta clase es una subclase de `PreparedStatement` y proporciona la misma funcionalidad. Un procedimiento almacenado puede contener múltiples sentencias SQL, o una serie de sentencias SQL —por tanto, el resultado podrían ser muchos objetos `ResultSet` diferentes—. A continuación se ilustra el caso en el que el resultado del procedimiento almacenado es un único `ResultSet`.

```
CallableStatement cst=
    con.prepareCall(" {call MuestraNumeroDePedidos}");
ResultSet rs = cst.executeQuery()
while (rs.next())
    ...
    ...
```

Llamada a procedimientos almacenados desde SQLJ

El procedimiento almacenado ‘`MuestraNumeroDePedidos`’ se llama del siguiente modo utilizando SQLJ:

```
// creación de la clase cursor
#sql Iterator InfoCliente(int idc, String nombrec, int contador);

// creación del cursor
InfoCliente infocliente;

// llamada al procedimiento almacenado
#sql infocliente = {CALL MuestraNumeroDePedidos};
while (infocliente.next()) {
    System.out.println(infocliente.idc() + "," +
        infocliente.contador());
}
```

6.5.3 SQL/PSM

Los sistemas de base de datos más importantes proporcionan la posibilidad de escribir procedimientos almacenados en un lenguaje simple y de propósito general muy próximo a SQL. En este apartado se estudia el estándar SQL/PSM, que es representativo de la mayor parte de los lenguajes específicos de bases de datos comerciales. En PSM se definen **módulos**, que son colecciones de procedimientos almacenados, relaciones temporales y otras declaraciones.

Un procedimiento almacenado se declara en SQL/PSM de la forma siguiente:

```
CREATE PROCEDURE nombre (parámetro1,..., parámetroN)
    declaraciones de variables locales
    código del procedimiento;
```

Se puede declarar una función del mismo modo como sigue:

```
CREATE FUNCTION nombre (parámetro1,..., parámetroN)
    RETURNS tipoDatosSql
    declaraciones de variables locales
    código de la función;
```

Cada parámetro es una terna compuesta por el modo (IN, OUT o INOUT como se comentó en el apartado anterior), el nombre del parámetro y el tipo de datos SQL del parámetro. Se pueden ver procedimientos SQL/PSM muy simples en el Apartado 6.5.1. En este caso, la declaración de variables locales estaba vacía, y el código del procedimiento estaba formado por una consulta SQL.

Se comenzará con un ejemplo de función SQL/PSM que ilustra las construcciones más importantes de SQL/PSM. La función toma como entrada un cliente identificado por su *idc* y un año. La función devuelve la valoración del cliente, definida como sigue: los clientes que han comprado más de diez libros durante el año tienen una valoración “dos”; los clientes que han comprado entre 5 y 10 libros tienen una valoración “uno”, y en cualquier otro caso tienen valoración “cero”. El siguiente código SQL/PSM calcula la valoración de un cliente para un año determinado.

```
CREATE PROCEDURE ClasificaCliente
    (IN idCliente INTEGER, IN año INTEGER)
    RETURNS INTEGER
DECLARE clasificación INTEGER;
DECLARE númPedidos INTEGER;
SET númPedidos =
    (SELECT COUNT(*) FROM Pedidos P WHERE P.idc = idCliente);
IF (númPedidos>10) THEN clasificación=2;
ELSEIF (númPedidos>5) THEN clasificación=1;
ELSE clasificación=0;
END IF;
RETURN clasificación;
```

Se usará este ejemplo para dar una breve visión general de algunas construcciones SQL/PSM:

- Se pueden declarar variables locales con la sentencia **DECLARE**. En el ejemplo, se declaran dos variables locales: “clasificación” y “númPedidos”.
- Las funciones SQL/PSM devuelven valores mediante la sentencia **RETURN**. En nuestro ejemplo, se devuelve el valor de la variable local “clasificación”.
- Se pueden asignar valores a variables con la sentencia **SET**. En el ejemplo anterior, se asigna el valor de retorno de una consulta a la variable “númPedidos”.
- SQL/PSM tiene bifurcaciones condicionales y bucles. Las bifurcaciones tienen la siguiente estructura:

```
IF (condición) THEN sentencias;
ELSEIF sentencias;
...
ELSEIF sentencias;
ELSE sentencias; END IF
```

Los bucles son de la forma

```
LOOP
    sentencias;
END LOOP
```

- Se pueden utilizar consultas como parte de expresiones en bifurcaciones condicionales; las consultas que devuelven un único valor pueden asignarse a variables como en el ejemplo anterior.
- Se pueden utilizar las mismas sentencias de cursor que en SQL incorporado (**OPEN**, **FETCH**, **CLOSE**), pero no son necesarias las construcciones **EXEC SQL**, y las variables no deben estar precedidas por un símbolo de dos puntos “:”.

Se ha realizado una muy breve visión de SQL/PSM; las referencias al final del capítulo proporcionan más información.

6.6 CASO DE ESTUDIO: LA LIBRERÍA EN INTERNET

TiposBD terminaron el diseño lógico de la base de datos, como se vio en el Apartado 3.8, y ahora consideran las consultas que tienen que soportar. Esperan que la lógica de la aplicación esté implementada en Java y por ello consideran que JDBC y SQLJ son candidatos para interconectar el sistema de base de datos con el código de la aplicación.

Recuérdese que TiposBD decidieron el siguiente esquema:

```
Libros(isbn: CHAR(10), título: CHAR(8), autor: CHAR(80),
       stock: INTEGER, precio: REAL, año_publicación: INTEGER)
Clientes(idc: INTEGER, nombrer: CHAR(80), dirección: CHAR(200))
Pedidos(númpedido: INTEGER, isbn: CHAR(10), idc: INTEGER,
        tarjeta: CHAR(16), cantidad: INTEGER, fecha_pedido: DATE, fecha_envío: DATE)
```

Ahora TiposBD consideran los tipos de consultas y actualizaciones que se producirán. Primero crean una lista de las tareas que realizará la aplicación. Entre las tareas realizadas por los clientes se encuentra las siguientes:

- Los clientes buscan libros por nombre de autor, título o ISBN.
- Los clientes se registran en el sitio Web. Los clientes registrados pueden querer cambiar su información de contacto. TiposBD se dan cuenta de que tienen que extender la tabla Clientes con información adicional para capturar la información de inicio de sesión y clave para cada cliente; este aspecto no se comentará con más profundidad.
- Los clientes deben pasar por caja una cesta de la compra para completar una venta.
- Los clientes añaden y eliminan libros de una “cesta de la compra” en el sitio Web.
- Los clientes comprueban el estado de los pedidos existentes y examinan pedidos antiguos.

Las tareas administrativas que tienen que realizar los empleados de B&N se listan a continuación.

- Los empleados buscan la información de contacto de clientes.
- Los empleados añaden libros nuevos al inventario.
- Los empleados realizan pedidos y necesitan actualizar la fecha de envío de libros individuales.
- Los empleados analizan los datos para encontrar clientes rentables y clientes que podrían responder a campañas especiales de marketing.

A continuación, TiposBD consideran los tipos de consultas que se producirán a partir de estas tareas. Para soportar las búsquedas de libros por nombre, autor, título o ISBN, TiposBD deciden escribir un procedimiento almacenado como el siguiente:

```
CREATE PROCEDURE BuscarPorISBN (IN isbn_libro CHAR(10))
    SELECT B.título, B.autor, B.stock, B.precio, B.año_publicación
    FROM Libros L
    WHERE L.isbn = isbn_libro
```

Hacer un pedido implica insertar uno o más registros en la tabla Pedidos. Como TiposBD no han elegido todavía la tecnología basada en Java para programar la lógica de la aplicación, asumen por ahora que los libros individuales en el pedido se almacenan en el nivel de aplicación en un array Java. Para finalizar el pedido, escriben el siguiente código JDBC mostrado en la Figura 6.11, que inserta los elementos del array en la tabla Pedidos. Nótese que este fragmento de código supone que se han asignado previamente algunas variables Java.

TiposBD escriben más código JDBC y procedimientos almacenados para todas las demás tareas, de forma similar a algunos de los fragmentos de código vistos en este capítulo.

- Establecimiento de una conexión a la base de datos, como se muestra en la Figura 6.2.
- Adición de libros nuevos al inventario, como se muestra en la Figura 6.3.

```

String sql = "INSERT INTO Pedidos VALUES(?, ?, ?, ?, ?, ?)";
PreparedStatement pst = con.prepareStatement(sql);
con.setAutoCommit(false);

try {
    // listaPedidos es un vector de objetos Pedido
    // númPedido es el número de pedido actual
    // idc es el ID del cliente
    // tarjeta es el número de tarjeta de crédito
    for (int i=0; i<listaPedidos.length(); i++)
        // Instanciación de los parámetros con valores
        Pedido pedidoActual = listaPedidos[i];
        pst.clearParameters();
        pst.setInt(1, númPedido);
        pst.setString(2, pedidoActual.getIsbn());
        pst.setInt(3, idc);
        pst.setString(4, tarjeta);
        pst.setInt(5, pedidoActual.getQty());
        pst.setDate(6, null);

        pst.executeUpdate();
    }
    con.commit();
} catch (SQLException e){
    con.rollback();
    System.out.println(e.getMessage());
}
}

```

Figura 6.11 Inserción de un Pedido en firme en la base de datos

- Procesamiento de resultados de consultas SQL como las de la Figura 6.4.
- Para cada cliente, mostrar cuántos pedidos se han realizado. Se muestra un procedimiento almacenado de ejemplo para esta consulta en la Figura 6.8.
- Incremento del número de copias disponibles de un libro en el inventario, como se muestra en la Figura 6.9.
- Clasificación de los clientes de acuerdo a sus compras, como en la Figura 6.10.

TiposBD tienen cuidado de hacer la aplicación robusta, procesando excepciones y avisos, como se muestra en la Figura 6.6.

TiposBD también deciden escribir un disparador, que se muestra en la Figura 6.12. Cada vez que un pedido nuevo se introduce en la tabla Pedidos, se asigna a fecha_envío el valor

NULL. El disparador procesa cada fila del pedido y llama al procedimiento almacenado “ActualizaFechaEnvío”. Este procedimiento (cuyo código no se muestra aquí) actualiza fecha_envío (prevista) para el pedido a “mañana” si stock del libro correspondiente en la tabla Libros es mayor que cero. En caso contrario, el procedimiento almacenado asigna a fecha_envío dos semanas después de la realización del pedido.

```
CREATE TRIGGER actualiza_FechaEnvío
    AFTER INSERT ON Pedidos
    /* Evento */
    FOR EACH ROW
    BEGIN CALL ActualizaFechaEnvío(new); END
    /* Acción */
```

Figura 6.12 Disparador para actualizar la fecha de envío de pedidos nuevos

6.7 PREGUNTAS DE REPASO

Las respuestas a las preguntas de repaso pueden encontrarse en los apartados indicados.

- ¿Por qué no es directa la integración de consultas SQL con un lenguaje de programación anfitrión? (**Apartado 6.1.1**)
- ¿Cómo se declaran variables en SQL incorporado? (**Apartado 6.1.1**)
- ¿Cómo se usan sentencias SQL en un lenguaje anfitrión? ¿Cómo se comprueban errores en la ejecución de una sentencia? (**Apartado 6.1.1**)
- Explíquese el desajuste de impedancia entre un lenguaje anfitrión y SQL, y describáse cómo los cursores pueden resolver este problema. (**Apartado 6.1.2**)
- ¿Qué propiedades pueden tener los cursores? (**Apartado 6.1.2**)
- ¿Qué es SQL dinámico y qué diferencias tiene respecto a SQL incorporado? (**Apartado 6.1.3**)
- ¿Qué es JDBC y cuáles son sus ventajas? (**Apartado 6.2**)
- ¿Qué son los componentes de la arquitectura JDBC? Describánse cuatro arquitecturas alternativas para los controladores JDBC. (**Apartado 6.2.1**)
- ¿Cómo se cargan los controladores JDBC en el código Java? (**Apartado 6.3.1**)
- ¿Cómo se gestionan las conexiones a fuentes de datos? ¿Qué propiedades pueden tener las conexiones? (**Apartado 6.3.2**)
- ¿Qué alternativas proporciona JDBC para ejecutar sentencias SQL DML y DDL? (**Apartado 6.3.3**)
- ¿Cómo se manejan excepciones y avisos en JDBC? (**Apartado 6.3.5**)

- ¿Qué funcionalidad proporciona la clase `DatabaseMetaData`? (**Apartado 6.3.6**)
- ¿Qué es SQLJ y qué diferencias presenta respecto a JDBC? (**Apartado 6.4**)
- ¿Por qué son importantes los procedimientos almacenados? ¿Cómo se declaran procedimientos almacenados y cómo se llaman desde el código de aplicación? (**Apartado 6.5**)

EJERCICIOS

Ejercicio 6.1 Contéstese brevemente a las siguientes preguntas.

1. Explíquense los siguientes términos: cursor, SQL incorporado, JDBC, SQLJ y procedimiento almacenado.
2. ¿Cuáles son las diferencias entre JDBC y SQLJ? ¿Por qué existen ambas alternativas?
3. Explíquense el término *procedimiento almacenado* y dense ejemplos de por qué son útiles.

Ejercicio 6.2 Explíquese cómo se realizan los siguientes pasos en JDBC:

1. Conectar a una fuente de datos.
2. Iniciar, comprometer y cancelar transacciones.
3. Llamar a un procedimiento almacenado.

¿Cómo se realizan estos pasos en SQLJ?

Ejercicio 6.3 Compárese el manejo de excepciones y avisos en SQL incorporado, SQL dinámico, JDBC y SQLJ.

Ejercicio 6.4 Contéstese a las siguientes preguntas.

1. ¿Por qué es necesario un precompilador para traducir SQL incorporado y SQLJ? ¿Por qué no se necesita para JDBC?
2. SQLJ y SQL incorporado utilizan variables del lenguaje anfitrión para pasar parámetros a consultas SQL, mientras que JDBC utiliza parámetros de sustitución marcados con un símbolo ‘?’ . Explíquese la diferencia y por qué se necesitan estos mecanismos.

Ejercicio 6.5 Un sitio Web dinámico genera páginas HTML con información almacenada en una base de datos. Cuando se solicita una página, se compone dinámicamente a partir de datos estáticos y datos en una base de datos, resultando en un acceso a base de datos. La conexión a la base de datos normalmente es un proceso costoso, pues es necesario asignar recursos y autenticar a los usuarios. Por tanto, una **cola de conexiones** —configurar una cola de conexiones de base de datos persistentes y después reutilizarlas para diferentes peticiones— puede mejorar significativamente el rendimiento de sitios Web basados en bases de datos. Como los servlets pueden mantener más información que simples peticiones, es posible crear una cola de conexiones y asignar recursos para las peticiones nuevas.

Escríbase una clase cola de conexiones que proporcione los siguientes métodos:

- Crear la memoria con un número determinado de conexiones abiertas en el sistema de base de datos.
- Obtener una conexión abierta de la cola.
- Liberar una conexión de la cola.
- Destruir la cola y cerrar todas las conexiones.

EJERCICIOS BASADOS EN PROYECTOS

En los siguientes ejercicios habrá que crea aplicaciones basadas en bases de datos. En este capítulo se crearán los componentes de la aplicación que acceden a la base de datos. En el próximo capítulo se extenderá este código a otros aspectos de la aplicación. Se puede encontrar información detallada sobre estos ejercicios y material de otros ejercicios en:

<http://www.cs.wisc.edu/~dbbook>

Ejercicio 6.6 Recuérdese la base de datos de Discos Sinpueblo (Notown Records) con la que trabajó en los Ejercicios 2.5 y 3.15. Ahora se pide diseñar un sitio Web para Sinpueblo. Debería proporcionar la siguiente funcionalidad:

- Los usuarios pueden buscar registros por nombre del músico, el título del álbum y el nombre de la canción.
- Los usuarios pueden registrarse en el sitio, y los usuarios registrados pueden iniciar sesión en el sitio. Una vez iniciada, los usuarios no deberían tener que iniciar sesión de nuevo a no ser que estén inactivos un largo periodo de tiempo.
- Los usuarios que han iniciado sesión en el sitio pueden añadir artículos en una cesta de compra.
- Los usuarios con artículos en sus cestas de compra pueden pasar por caja y realizar una compra.

Sinpueblo desea utilizar JDBC para acceder a la base de datos. Escriba código JDBC que realice los accesos y manipulaciones de datos necesarias. Este código se integrará con la lógica de aplicación y presentación en el próximo capítulo.

Si Sinpueblo hubiera elegido SQLJ en lugar de JDBC, ¿cómo cambiaría su código?

Ejercicio 6.7 Recuérdese el esquema de base de datos para Recetas-R-X que se creó en el Ejercicio 2.7. La cadena de farmacias de Recetas-R-X le ha contratado para diseñar su nuevo sitio Web. El sitio Web tiene dos clases de usuarios: doctores y pacientes. Los doctores deben poder introducir nuevas recetas para sus pacientes y modificar las recetas existentes. Los pacientes deben poder identificarse como pacientes de un doctor; deben poder comprobar el estado de sus recetas en línea; y deben poder comprar las recetas en línea de forma que se les envíen los medicamentos a sus domicilios.

Llévense a cabo los mismos pasos del Ejercicio 6.6 para escribir código JDBC que realice los accesos y manipulaciones necesarias de datos. Este código se integrará con la lógica de la aplicación y presentación en el próximo capítulo.

Ejercicio 6.8 Recuérdese el esquema de base de datos de la universidad del Ejercicio 5.1. La universidad ha decidido pasar la inscripción de alumnos a un sistema en línea. El sitio Web tiene dos clases de usuarios: docentes y alumnos. Los docentes deben poder crear nuevos cursos y eliminar los cursos existentes, y los alumnos deben poder inscribirse en cursos existentes.

Llévense a cabo los mismos pasos que en el Ejercicio 6.6 para escribir código JDBC que realice los accesos y manipulaciones necesarias de datos. Este código se integrará con la lógica de la aplicación y presentación en el próximo capítulo.

Ejercicio 6.9 Recuérdese el esquema de reserva de vuelos en el que trabajó en el Ejercicio 5.3. Diseñe un sistema de reserva de vuelos en línea. El sistema de reservas tendrá dos tipos de usuarios: empleados de líneas aéreas y pasajeros. Los empleados pueden programar nuevos vuelos y cancelar los vuelos existentes. Los pasajeros pueden reservar vuelos existentes para un destino dado.

Llévense a cabo los mismos pasos que en el Ejercicio 6.6 para escribir código JDBC que realice los accesos y manipulaciones necesarias de datos. Este código se integrará con la lógica de la aplicación y presentación en el próximo capítulo.



NOTAS BIBLIOGRÁFICAS

Se puede encontrar información sobre ODBC en la página Web de Microsoft (www.microsoft.com/data/odbc), y sobre JDBC en la página Web de Java (java.sun.com/products/jdbc). Existen muchos libros sobre ODBC, por ejemplo la Guía del desarrollador ODBC de Sander [388] y el SDK de ODBC de Microsoft [333]. Los libros sobre JDBC incluyen trabajos de Hamilton *et al.* [233], Reese [373] y White *et al.* [470].



7

APLICACIONES DE INTERNET

- ➔ ¿Cómo se nombran los recursos en Internet?
- ➔ ¿Cómo pueden comunicarse entre sí los navegadores y los servidores Web?
- ➔ ¿Cómo se pueden presentar documentos en Internet? ¿Cómo se puede diferenciar entre formato y contenido?
- ➔ ¿Qué es una arquitectura de aplicaciones de tres capas? ¿Cómo se escriben aplicaciones de tres capas?
- ➔ ¿Por qué existen los servidores de aplicaciones?
- ➡ **Conceptos clave:** identificadores uniformes de recursos (Uniform Resource Identifiers, URI), localizadores uniformes de recursos (Uniform Resource Locators, URL); protocolo de transferencia de hipertexto (Hypertext Transfer Protocol, HTTP), protocolo sin estado; Java; HTML; XML, DTD de XML; arquitectura de tres capas, arquitectura cliente-servidor; formularios HTML; JavaScript; hojas de estilo en cascada, XSL; servidor de aplicaciones; interfaz de pasarela común (Common Gateway Interface, CGI); servlet; páginas de servidor de Java (JavaServer Pages, JSP); cookie.

¡Guau! ¡Ahora han conseguido Internet en los computadores!

— Homer Simpson, *Los Simpsons*

7.1 INTRODUCCIÓN

La proliferación de redes de ordenadores, incluyendo Internet e “intranets” corporativas, ha permitido a los usuarios acceder a un gran número de fuentes de datos. Este incremento en el acceso a bases de datos probablemente tendrá un gran impacto práctico; ahora se pueden ofrecer datos y servicios directamente a los clientes de maneras hasta hace poco imposibles. Ejemplos de tales aplicaciones de **comercio electrónico** incluyen la compra de

libros mediante un minorista Web como Amazon.com, subastas en línea en sitios como eBay e intercambio de ofertas y especificaciones de productos entre compañías. La aparición de estándares como XML para describir el contenido de documentos probablemente acelerará el comercio electrónico y otras aplicaciones en línea.

Mientras que la primera generación de sitios en Internet eran colecciones de archivos HTML, la mayor parte de los sitios importantes hoy en día almacenan una gran parte de sus datos (si no todos) en sistemas de bases de datos. Éstos dependen de los SGBD para proporcionar respuestas rápidas y fiables a las peticiones de usuarios recibidas a través de Internet. Esto es especialmente cierto en sitios de comercio electrónico y otras aplicaciones empresariales.

En este capítulo se presenta una visión general de los conceptos centrales del desarrollo de aplicaciones de Internet. Se comienza con una visión básica de cómo funciona Internet en el Apartado 7.2. Se introducirán HTML y XML, dos formatos de datos que se utilizan para presentar datos en Internet, en los Apartados 7.3 y 7.4. En el Apartado 7.5 se explican las arquitecturas de tres capas, una forma de estructurar aplicaciones de Internet en diferentes niveles que encapsulan distintas funcionalidades. En los Apartados 7.6 y 7.7 se describen en detalle las capas de presentación e intermedia; el SGBD es la tercera capa. Concluiremos el capítulo tratando el ejemplo B&N en el Apartado 7.8.

Los ejemplos que aparecen en este capítulo están disponibles en línea en:

<http://www.cs.wisc.edu/~dbbook>

7.2 CONCEPTOS DE INTERNET

Internet ha aparecido como un conector universal entre sistemas software globalmente distribuidos. Para entender cómo funciona, se empezará estudiando dos temas básicos: cómo se identifican los sitios en Internet, y cómo se pueden comunicar los programas de un sitio con otros sitios.

Primero se introducen los identificadores uniformes de recursos (Uniform Resource Identifiers, URI), un esquema de nombres para localizar recursos en Internet, en el Apartado 7.2.1. Después se estudiará del protocolo más popular para acceder a recursos en Web, el protocolo de transferencia de hipertexto (hypertext transfer protocol, HTTP) en el Apartado 7.2.2.

7.2.1 Identificadores uniformes de recursos (URI)

Los identificadores uniformes de recursos (**Uniform Resource Identifiers, URI**) son cadenas de caracteres que identifican únicamente recursos en Internet. Un **recurso** es cualquier clase de información que se pueda identificar con un URI, como por ejemplo páginas Web, imágenes, archivos descargables, servicios que se puedan invocar remotamente, cuentas de correo electrónico, etcétera. La clase más común de recurso es un archivo estático (como por ejemplo un documento HTML), pero también puede ser un recurso un archivo HTML generado dinámicamente, una película, la salida de un programa, etc.

Un URI tiene tres partes:

- El (nombre del) protocolo utilizado para acceder al recurso.



Aplicaciones distribuidas y arquitecturas orientadas a servicios. El advenimiento de XML, debido a su naturaleza débilmente acoplada, ha hecho posible el intercambio de información entre diferentes aplicaciones en una medida nunca vista antes. Usando XML para el intercambio de información, se pueden escribir aplicaciones en diferentes lenguajes de programación, ejecutarlas en distintos sistemas operativos, y aun así poder compartir información entre ellas. También existen estándares para describir externamente el contenido previsto de un archivo o mensaje XML, especialmente en el estándar W3C de esquemas XML recientemente adoptado.

Un concepto prometedor que ha surgido de la revolución XML es la noción de **servicio Web**. Un servicio Web es una aplicación que proporciona un servicio bien definido, empaquetado como un conjunto de procedimientos invocables remotamente y accesible a través de Internet. Los servicios Web tienen el potencial para crear nuevas aplicaciones *componiendo* servicios Web existentes —todos ellos comunicándose sin problemas gracias al intercambio de información estandarizado XML—. Se han desarrollado o están actualmente en desarrollo diversas tecnologías que facilitan el diseño e implementación de aplicaciones distribuidas. SOAP es un estándar W3C para la invocación remota de servicios basados en XML (considérese RPC XML) que permite la comunicación entre aplicaciones distribuidas mediante mensajes XML estructurados y con tipos. Las llamadas SOAP pueden viajar sobre diversas capas de transporte, incluyendo HTTP (que en parte está haciendo que SOAP tenga tanto éxito) y otras capas fiables de paso de mensajes. En relación con el estándar SOAP, de W3C están el lenguaje de descripción de servicios Web (**Web Services Description Language, WSDL**) para describir interfaces de servicios Web, y la **descripción, descubrimiento e integración universal** (Universal Description, Discovery, and Integration, UDDI), un registro estándar de servicios Web basado en WSDL (una especie de páginas amarillas para servicios Web).

Los servicios Web basados en SOAP son la base de la arquitectura **.NET** recientemente publicada por Microsoft, que forma su infraestructura de desarrollo de aplicaciones y sistema de ejecución asociado para desarrollar aplicaciones distribuidas, así como las propuestas de servicios Web de los fabricantes de software más importantes, como IBM, BEA y otros. Muchos grandes fabricantes de aplicaciones software (compañías importantes como PeopleSoft o SAP) han anunciado planes para proporcionar interfaces de servicios Web para sus productos y los datos que manejan, y muchos confían que XML y los servicios Web darán la respuesta al viejo problema de la integración de aplicaciones empresariales. Los servicios Web también se ven como la base natural para la siguiente generación de sistemas de gestión de procesos de negocio (o de flujos de trabajo).

- El ordenador donde está localizado el recurso.
- La vía de acceso del recurso dentro del ordenador donde está localizado.

Considérese un URI de ejemplo, como `http://www.tiendalibros.com/index.html`. Este URI se puede interpretar como sigue. Utiliza el protocolo HTTP (descrito en el siguiente apartado) para recuperar el documento `index.html` localizado en `www.tiendalibros.com`. Éste es un ejemplo de localizador universal de recursos (**Universal Resource Locator, URL**), un subconjunto del esquema más general de nombrado URI; la distinción no es importante para nuestro objetivo. Otro ejemplo es el siguiente fragmento de HTML que muestra un URI que es una dirección de correo electrónico:

```
<a href="mailto:webmaster@bookstore.com">Enviar un email al webmaster.</A>
```

7.2.2 El protocolo de transferencia de hipertexto (HTTP)

Un **protocolo de comunicación** es un conjunto de estándares que define la estructura de los mensajes entre dos partes que se comunican, de forma que puedan entender mutuamente los mensajes. El **protocolo de transferencia de hipertexto** (Hypertext Transfer Protocol, HTTP) es el protocolo de comunicación más comúnmente usado en Internet. Es un protocolo cliente-servidor en el que un cliente (normalmente un navegador Web) envía una petición a un servidor HTTP, que a su vez envía una respuesta al cliente. Cuando un usuario solicita una página Web (por ejemplo, pulsa con el ratón sobre un hipervínculo), el navegador envía **mensajes de petición HTTP** de los objetos en la página al servidor. El servidor recibe las peticiones y responde con **mensajes de respuesta HTTP**, que incluyen los objetos solicitados. Es importante darse cuenta de que HTTP se utiliza para transmitir cualquier tipo de recurso, no solamente archivos, pero la mayor parte de los recursos en Internet hoy en día son archivos estáticos o bien archivos resultado de programas del servidor.

Una variante del protocolo HTTP denominada **capa de sockets seguros** (Secure Sockets Layer, SSL) utiliza cifrado para intercambiar información con seguridad entre cliente y servidor. Se pospone el tratamiento de SSL al Apartado 14.5.2 y se presenta el protocolo HTTP básico en este capítulo.

Como ejemplo se estudiará lo que ocurre si un usuario pulsa con el ratón en el siguiente enlace: `http://www.tiendalibros.com/index.html`. En primer lugar se explica la estructura de un mensaje de petición HTTP y después la estructura de un mensaje de respuesta HTTP.

Solicitudes HTTP

El cliente (navegador Web) establece una conexión con el servidor Web que aloja el recurso y envía un mensaje de solicitud HTTP. El siguiente ejemplo muestra un mensaje de petición HTTP:

```
GET index.html HTTP/1.1
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
```

La estructura general de una solicitud HTTP consiste en varias líneas de texto ASCII, con una línea en blanco al final. La primera línea, la **línea de solicitud**, tiene tres campos: el **campo método HTTP**, el **campo URI** y el **campo versión HTTP**. El campo **método** puede tomar los valores **GET** o **POST**; en el ejemplo el mensaje solicita el objeto **index.html**. (Se comentarán en detalle las diferencias entre **GET** y **POST** en el Apartado 7.11.) El campo versión indica la versión de HTTP que utiliza el cliente y que se puede usar para futuras extensiones del protocolo. El **agente de usuario** indica el tipo de cliente (por ejemplo, versiones de Netscape o de Internet Explorer); no se comentará esta opción en más profundidad. La tercera línea, que comienza con **Accept**, indica los tipos de archivos que puede aceptar el cliente. Por ejemplo, si la página **index.html** contiene un archivo de película con la extensión **.mpg**, el servidor no lo enviará al cliente, pues éste no está preparado para aceptarlo.

Respuestas HTTP

El servidor contesta con un mensaje de **respuesta**. Recupera la página **index.html**, la utiliza para ensamblar el mensaje de respuesta HTTP y envía el mensaje al siguiente. Un ejemplo de respuesta HTTP es el siguiente:

```
HTTP/1.1 200 OK
Date: Mon, 04 Mar 2002 12:00:00 GMT
Content-Length: 1024
Content-Type: text/html
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
<HTML>
<HEAD>
</HEAD>
<BODY>
<H1>Librería en Internet Benito y Norberto</H1>
Nuestro inventario:
<H3>Ciencia</H3>
<B>El carácter de la ley física</B>
...

```

El mensaje de respuesta HTTP tiene tres partes: una línea de estado, varias líneas de cabecera y el cuerpo del mensaje (que contiene el objeto que el cliente pidió). La **línea de estado** tiene tres campos (análogos a la línea de solicitud del mensaje de solicitud HTTP): la versión de HTTP (**HTTP/1.1**), un código de estado (200) y un mensaje del servidor (**OK**). Algunos códigos de estado y mensajes comunes son:

- 200 **OK**. La solicitud tuvo éxito y el objeto está contenido en el cuerpo del mensaje de respuesta.
- 400 **Bad Request**. Código de error genérico que indica que el servidor no pudo realizar la petición.
- 404 **Not Found**. El objeto solicitado no existe en el servidor.

- 505 HTTP Version Not Supported. La versión del protocolo HTTP que utiliza el cliente no está soportada por el servidor. (Recuérdese que la versión del protocolo está incluida en la solicitud del cliente.)

Nuestro ejemplo tiene tres **líneas de cabecera**: la línea de cabecera **Date** indica la hora y la fecha cuando se creó la respuesta HTTP (no es la fecha de creación del objeto). La línea de cabecera **Last-Modified** corresponde al momento en que se creó el objeto solicitado. La línea **Content-Length** contiene el número de bytes del objeto que se envía después de la última línea de cabecera. La línea **Content-Type** indica que el objeto del mensaje es texto HTML.

El cliente (el navegador Web) recibe el mensaje de respuesta, extrae el archivo HTML, lo analiza sintácticamente y lo muestra. Al hacerlo, puede encontrar otros URI en el archivo, y entonces utiliza el protocolo HTTP para obtener cada uno de esos recursos, estableciendo una conexión nueva cada vez.

Un aspecto importante es que HTTP es un protocolo **sin estado**. Cada mensaje —del cliente al servidor HTTP y viceversa— es autónomo, y la conexión que se establece en una solicitud se mantiene solamente hasta que se envía el mensaje de respuesta. El protocolo no proporciona ningún mecanismo para “recordar” interacciones anteriores entre el cliente y el servidor.

Esta característica del protocolo HTTP tiene un impacto fundamental en cómo se escriben las aplicaciones de Internet. Considérese un usuario que interactúa con la aplicación de librería. Supóngase que la librería permite a los usuarios iniciar una sesión en el sitio y realizar varias acciones como pedir libros o cambiar sus direcciones, sin necesidad de iniciar sesión de nuevo (hasta que expire la sesión o el usuario la finalice). ¿Cómo se puede saber si un usuario ha iniciado su sesión o no? Como HTTP no tiene estado, no se puede cambiar a otro estado diferente (como un estado “sesión iniciada”) en el nivel del protocolo. En su lugar, para cada solicitud que el usuario (o más precisamente, su navegador Web) envía al servidor, se debe codificar cualquier información de *estado* que requiera la aplicación, como, por ejemplo, el estado de inicio de sesión. Como alternativa, las aplicaciones del lado del servidor pueden mantener esta información de estado y consultarla por cada solicitud. Se verá más a fondo este tema en el Apartado 7.7.5.

Nótese que la falta de estado de HTTP es un equilibrio entre la facilidad de implementación del protocolo HTTP y la simplicidad del desarrollo de aplicaciones. Los diseñadores de HTTP eligieron mantener el protocolo simple y postergaron cualquier funcionalidad más allá de la solicitud de objetos a niveles de aplicación por encima del protocolo HTTP.

7.3 DOCUMENTOS HTML

Este apartado y el siguiente se centrarán en introducir HTML y XML. En el Apartado 7.6 se verá la forma en que las aplicaciones pueden utilizar HTML y XML para crear formularios que capturen la entrada del usuario, se comuniquen con un servidor HTTP y conviertan los resultados producidos por la capa de gestión de datos en uno de estos formatos.

HTML es un lenguaje simple que se utiliza para describir un documento. También se denomina **lenguaje de marcado** porque HTML funciona extendiendo el texto normal con “marcas” que tienen un significado especial para un navegador Web. Los comandos del lenguaje, llamados **etiquetas** (normalmente) consisten en una **etiqueta de inicio** y una **etiqueta**

de fin de la forma <ETIQUETA> y </ETIQUETA>, respectivamente. Por ejemplo, considérese el fragmento de HTML mostrado en la Figura 7.1. Describe una página Web que muestra una lista de libros. El documento está entre las etiquetas <HTML> y </HTML>, que lo marcan como un documento HTML. El resto del documento —dentro de <BODY> ... </BODY>— contiene información sobre tres libros. Los datos de cada libro se presentan como una lista no ordenada (*UL*, *unordered list*) cuyas entradas están marcadas con la etiqueta *LI*. HTML define el conjunto de etiquetas válidas así como su significado. Por ejemplo, HTML especifica que la etiqueta <TITLE> es una etiqueta válida que denota el título del documento. Otro ejemplo: la etiqueta siempre representa una lista no ordenada.

```

<HTML>
<HEAD>
</HEAD>
<BODY>
<H1>Librería en Internet Benito y Norberto</H1>
Nuestro inventario:
<H3>Ciencia</H3>
<B>El carácter de la ley física</B>
<UL>
<LI>Author: Richard Feynman</LI>
<LI>Publicado en 2000</LI>
<LI>Rústica</LI>
</UL>
<H3>Ficción</H3>
<B>Esperando a Mahatma</B>
<UL>
<LI>Author: R.K. Narayan</LI>
<LI>Publicado en 1955</LI>
</UL>
<B>El profesor de inglés</B>
<UL>
<LI>Author: R.K. Narayan</LI>
<LI>Publicado en 1945</LI>
<LI>Tapa dura</LI>
</UL>
</BODY>
</HTML>
```

Figura 7.1 Listado de libros en HTML

En los documentos HTML se puede incluir audio, vídeo e incluso programas (escritos en Java, un lenguaje altamente portable). Cuando un usuario recupera un documento con el navegador apropiado, se muestran las imágenes del documento, se reproducen los archivos de

audio y vídeo, y los programas incluidos se ejecutan en la máquina del usuario; el resultado es una completa presentación multimedia. La facilidad con la que se pueden crear —existen editores visuales que generan HTML— y acceder a documentos HTML utilizando navegadores de Internet ha impulsado el crecimiento explosivo de la Web.

7.4 DOCUMENTOS XML

En este apartado se introducirá el formato de documentos XML y se verá la forma de utilizarlo en las aplicaciones. La gestión de documentos XML en un SGBD plantea nuevos desafíos; este aspecto se comentará de XML en el Capítulo 18.

Aunque HTML se puede utilizar para mostrar documentos, no es adecuado para describir la estructura del contenido de aplicaciones más generales. Por ejemplo, se puede enviar el documento HTML mostrado en la Figura 7.1 a otra aplicación que lo muestre, pero esta aplicación no puede distinguir los nombres de los autores de sus apellidos. (La aplicación puede intentar recuperar esta información examinando en el texto dentro de las etiquetas, pero se pierde el objetivo de utilizar etiquetas para mostrar la estructura del documento.) Por ello, HTML no es adecuado para intercambiar documentos complejos con especificaciones de productos u ofertas, por ejemplo.

El **lenguaje de marcado extensible** (Extensible Markup Language, XML) es un lenguaje de marcado desarrollado para solucionar las deficiencias de HTML. En lugar de un conjunto de etiquetas cuyo significado está especificado por el lenguaje (como ocurre en HTML), XML permite a los usuarios definir conjuntos nuevos de etiquetas que se pueden utilizar para estructurar cualquier tipo de datos o documentos que el usuario deseé transmitir. XML es un puente importante entre la visión de los datos orientada a documentos implícita en HTML y la visión de los datos orientada a esquemas, lo cual es fundamental para un SGBD. Tiene el potencial para integrar más que nunca los sistemas de bases de datos en las aplicaciones Web.

XML apareció por la confluencia de dos tecnologías, SGML y HTML. El **lenguaje de marcado generalizado estándar** (Standard Generalized Markup Language, SGML) es un metalenguaje que permite definir lenguajes de definición de datos y documentos tales como HTML. El estándar SGML fue publicado en 1988, y lo han adoptado muchas organizaciones que manejan un gran número de documentos complejos. Debido a su generalidad, SGML es complejo y requiere programas sofisticados para aprovechar todo su potencial. XML se desarrolló para tener gran parte de la potencia de SGML a la vez que se mantiene relativamente simple. Sin embargo, XML, como SGML, permite la definición de nuevos lenguajes de marcado de documentos.

Aunque XML no impide al usuario diseñar etiquetas que codifiquen la visualización de datos en un navegador Web, existe un lenguaje de estilo para XML denominado **lenguaje de estilo extensible** (Extensible Style Language, XSL). XSL es una forma estándar para la descripción del modo en que debería visualizarse un documento que se adhiera a un determinado vocabulario de etiquetas.

Los objetivos de diseño de XML. XML comenzó a desarrollarse en 1996 por un grupo de trabajo bajo la orientación del Grupo de Interés Especial XML del World Wide Web Consortium (W3C). Entre los objetivos de diseño de XML se incluían los siguientes:

1. XML debe ser compatible con SGML.
2. Debe ser fácil escribir programas que procesen documentos XML.
3. El diseño de XML debe ser formal y conciso.

7.4.1 Introducción a XML

Se usará como ejemplo el pequeño documento XML mostrado en la Figura 7.2.

- **Elementos.** Los elementos, también denominados **etiquetas**, son los bloques constructores fundamentales de un documento XML. El principio de contenido de un elemento ELM está marcado con <ELM>, que se llama la **etiqueta de inicio**, y el final está marcado con </ELM>, denominado **etiqueta de fin**. En el documento de ejemplo, el elemento LISTALIBROS encierra toda la información del documento. El elemento LIBRO delimita todos los datos asociados a un solo libro. Los elementos XML son sensibles a las mayúsculas: el elemento LIBRO es diferente de Libro. Los elementos deben anidarse apropiadamente. Las etiquetas de inicio que aparecen dentro del contenido de otras etiquetas deben tener la correspondiente etiqueta de fin. Por ejemplo, considérese el siguiente fragmento XML:

```
<LIBRO>
  <AUTOR>
    <NOMBRE>Richard</NOMBRE>
    <APELIDO>Feynman</APELIDO>
  </AUTOR>
</LIBRO>
```

El elemento AUTOR está completamente anidado dentro del elemento LIBRO, y tanto APELLIDO como NOMBRE están dentro del elemento AUTOR.

- **Atributos.** Un elemento puede tener atributos descriptivos que proporcionen información adicional sobre el elemento. Los valores de los atributos se asignan dentro de la etiqueta de inicio del elemento. Por ejemplo, si ELM denota un elemento con el atributo att, se puede asignar el valor de att a valor mediante la siguiente expresión: <ELM att="valor">. Todos los valores de atributos deben estar entre comillas. En la Figura 7.2, el elemento LIBRO tiene dos atributos. El atributo GENERO contiene el género del libro (ciencia o ficción), y el atributo FORMATO indica si el formato del libro es rústica o de tapa dura.
- **Referencias de entidad.** Las entidades son atajos para fragmentos de texto o de contenido de archivos externos, y el uso de una entidad en un documento XML se denomina

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<LISTALIBROS>
<LIBRO GENERO="Ciencia" FORMATO="Rústica">
    <AUTOR>
        <NOMBRE>Richard</NOMBRE>
        <APELLIDO>Feynman</APELLIDO>
    </AUTOR>
    <TITULO>El carácter de la ley física</TITULO>
    <PUBLICADO>1980</PUBLICADO>
</LIBRO>
<LIBRO GENERO="Ficción">
    <AUTOR>
        <NOMBRE>R.K.</NOMBRE>
        <APELLIDO>Narayan</APELLIDO>
    </AUTOR>
    <TITULO>Esperando a Mahatma</TITULO>
    <PUBLICADO>1955</PUBLICADO>
</LIBRO>
<LIBRO GENERO="Ficción">
    <AUTOR>
        <NOMBRE>R.K.</NOMBRE>
        <APELLIDO>Narayan</APELLIDO>
    </AUTOR>
    <TITULO>El profesor de inglés</TITULO>
    <PUBLICADO>1945</PUBLICADO>
</LIBRO>
</LISTALIBROS>

```

Figura 7.2 Información de libros en XML

referencia de entidad. Siempre que aparece una referencia de entidad en un documento, se sustituye textualmente por su contenido. Las referencias de entidad comienzan con un símbolo “&” y terminan con “;”. XML tiene cinco entidades predefinidas para caracteres con significado especial en XML. Por ejemplo, el carácter < que marca el principio de un comando XML está reservado y debe representarse mediante la entidad 1t. Los otros cuatro caracteres reservados son &, >, ” y ‘, que están representados respectivamente por las entidades amp, gt, quot y apos. Por ejemplo, el texto ‘1 < 5’ tiene que codificarse en un documento XML como '1<5'. También es posible utilizar entidades para insertar caracteres Unicode arbitrarios en el texto. **Unicode** es un estándar para representar caracteres, similar a ASCII. Por ejemplo, es posible mostrar el carácter Hiragana japonés *a* utilizando la referencia de entidad あ.

- **Comentarios.** Se pueden insertar comentarios en cualquier parte de un documento XML. Deben comenzar con `<!--` y terminar con `-->`. Los comentarios pueden contener cualquier texto excepto la cadena de caracteres `--`.
- **Declaraciones de tipo de documento (DTD).** En XML es posible definir nuestro propio lenguaje de marcado. Una DTD es un conjunto de reglas que permiten especificar nuestro propio conjunto de elementos, atributos y entidades. De este modo, una DTD es básicamente una gramática que indica las etiquetas permitidas, el orden en que pueden aparecer y cómo se pueden anidar. En el siguiente apartado se comentarán las DTD en detalle.

Se dice que un documento XML está **bien formado** si no tiene una DTD asociada pero sigue estas directrices estructurales:

- El documento comienza con una declaración XML. Un ejemplo de declaración XML es la primera línea del documento XML mostrado en la Figura 7.2.
- Un **elemento raíz** contiene a todos los demás elementos. En el ejemplo, el elemento raíz es `LISTALIBROS`.
- Todos los elementos deben estar anidados correctamente. Esto quiere decir que las etiquetas de inicio y de fin de un elemento dado deben aparecer dentro del mismo elemento que los contiene.

7.4.2 DTD de XML

Una DTD es un conjunto de reglas que permite especificar nuestro propio conjunto de elementos, atributos y entidades. Una DTD especifica los elementos que pueden utilizarse y las restricciones que se aplican sobre ellos; por ejemplo, la forma en que pueden anidarse los elementos y el lugar del documento en que pueden aparecer estos elementos. Se dice que un documento es **válido** si se le asocia una DTD y el documento está estructurado de acuerdo a las reglas definidas por la DTD. En lo que resta de este apartado, se empleará la DTD de ejemplo mostrada en la Figura 7.3 para ilustrar la construcción de DTD.

Una DTD se define con `<!DOCTYPE nombre [declaraciónDTD]>`, donde `nombre` es el nombre de la etiqueta más externa, y `declaraciónDTD` es el texto de las reglas de la DTD. La DTD comienza con el elemento más externo —el *elemento raíz*— que es `LISTALIBROS` en el ejemplo. Considérese la siguiente regla:

```
<!ELEMENT LISTALIBROS (LIBRO)*>
```

Esta regla dice que el elemento `LISTALIBROS` está formado por cero o más elementos `LIBRO`. El `*` después de `LIBRO` indica cuántos elementos `LIBRO` pueden aparecer dentro del elemento `LISTALIBROS`. Un `*` denota cero o más apariciones, un `+` representa una o más apariciones y un `?` indica cero o una aparición. Por ejemplo, si se desea asegurar que una `LISTALIBROS` tenga al menos un libro, se puede cambiar la regla de la siguiente forma:

```
<!ELEMENT LISTALIBROS (LIBRO)+>
```

Considérese la siguiente regla:

```

<!DOCTYPE LISTALIBROS [
  <!ELEMENT LISTALIBROS (LIBRO)*>
    <!ELEMENT LIBRO (AUTOR,TITULO,PUBLICADO?)>
      <!ELEMENT AUTOR (NOMBRE,APELLIDO)>
        <!ELEMENT NOMBRE (#PCDATA)>
        <!ELEMENT APELLIDO (#PCDATA)>
      <!ELEMENT TITULO (#PCDATA)>
      <!ELEMENT PUBLICADO (#PCDATA)>
    <!ATTLIST LIBRO GENERO (Ciencia|Ficción) #REQUIRED>
    <!ATTLIST LIBRO FORMATO (Rústica|Tapa dura) "Rústica">
]>

```

Figura 7.3 DTD de XML de la librería

<!ELEMENT LIBRO (AUTOR,TITULO,PUBLICADO?)>

Esta regla estipula que un elemento LIBRO contiene un elemento AUTOR, un elemento TITULO y un elemento opcional PUBLICADO. Obsérvese el uso de ? para indicar que la información es opcional, teniendo cero o una aparición del elemento. Véase la siguiente regla:

<!ELEMENT APELLIDO (#PCDATA)>

Hasta ahora sólo se han considerado elementos que contuviesen otros elementos. Esta regla declara que APELLIDO es un elemento que no contiene otros elementos, sino que solamente contiene texto. Los elementos que sólo contienen otros elementos se dice que tienen **contenido de elementos**, mientras que los elementos que también contienen #PCDATA tienen **contenido mixto**. En general, una declaración de tipo de elemento tiene la siguiente estructura:

<!ELEMENT (tipoDeContenido)>

Hay cinco tipos de contenido posibles:

- Otros elementos.
- El símbolo especial #PCDATA, que indica datos de caracteres.
- El símbolo especial EMPTY, que indica que el elemento no tiene contenido. Los elementos que no tienen contenido no necesitan tener una etiqueta de fin.
- El símbolo especial ANY, que indica que se permite cualquier contenido. Este contenido debería evitarse siempre que sea posible, pues impide cualquier comprobación de la estructura del documento dentro del elemento.
- Una **expresión regular** construida a partir de los cuatro tipos anteriores. Una expresión regular es de la forma:
 - exp1, exp2, exp3. Una lista de expresiones regulares.
 - exp*. Una expresión opcional (cero o más apariciones).

- exp?. Una expresión opcional (cero o una aparición).
- exp+. Una expresión obligatoria (una o más apariciones).
- exp1 | exp2: exp1 o exp2.

Los atributos de los elementos se declaran fuera del elemento. Por ejemplo, considérese la siguiente declaración de atributo de la Figura 7.3:

```
<!ATTLIST LIBRO GENERO (Ciencia|Ficción) #REQUIRED>
```

Este fragmento de DTD de XML especifica el atributo **GENERO** del elemento **LIBRO**. El atributo puede tomar dos valores: Ciencia o Ficción. Cada elemento **LIBRO** debe describirse con un atributo **GENERO** en su etiqueta de inicio, pues este atributo es obligatorio, como se indica con **#REQUIRED**. Considérese la estructura general de una declaración de atributo de una DTD:

```
<!ATTLIST nombreElemento (nombreAtr tipoAtr especPorDefecto)+>
```

La palabra clave **ATTLIST** indica el principio de una declaración de atributo. La cadena de caracteres **nombreElemento** es el nombre del elemento con el que está asociada la definición de atributo. A continuación se declaran uno o más atributos. Cada atributo tiene un nombre, indicado con **nombreAtr**, y un tipo, indicado con **tipoAtr**. XML define varios tipos posibles para los atributos. Aquí se verá sólo el **tipo cadena de caracteres** y los **tipos enumerados**. Un atributo de tipo cadena puede tomar como valor cualquier cadena de caracteres. Se puede declarar un atributo de este tipo indicando en el campo el tipo **CDATA**. Por ejemplo, se puede declarar un tercer atributo de tipo cadena del elemento **LIBRO** como el siguiente:

```
<!ATTLIST LIBRO edición CDATA "1">
```

Si un atributo es de un tipo enumerado, deben listarse todos los valores posibles en la declaración del atributo. En el ejemplo, el atributo **GENERO** es de tipo enumerado; los valores que puede tomar son “Ciencia” y “Ficción”.

La última parte de una declaración de atributo es su **especificación predeterminada**. La DTD de la Figura 7.3 muestra dos especificaciones predeterminadas diferentes: **#REQUIRED** y la cadena “Rústica”. La especificación **#REQUIRED** indica que el atributo es obligatorio y que siempre que aparezca un elemento del tipo asociado en algún lugar del documento, debe especificarse un valor para el atributo. La especificación representada por la cadena “Rústica” indica que el atributo no es obligatorio; siempre que aparezca un elemento sin asignar un valor al atributo, éste automáticamente tomará el valor “Rústica”. Por ejemplo, se puede hacer que “Ciencia” sea el valor predeterminado del atributo **GENERO**:

```
<!ATTLIST LIBRO GENERO (Ciencia|Ficción) "Ciencia">
```

Respecto al ejemplo de la librería, en la Figura 7.4 se muestra el documento XML con una referencia a la DTD.

7.4.3 DTD de dominios específicos

Recientemente se han desarrollado DTD para diversos dominios especializados —incluyendo un amplio rango de dominios comerciales, de ingeniería, financieros, industriales— y gran

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE LISTALIBROS SYSTEM "libros.dtd">
<LISTALIBROS>
    <LIBRO GENERO="Ciencia" FORMATO="Rústica">
        <AUTOR>
            ...

```

Figura 7.4 Información de libros en XML

Esquema XML. El mecanismo de las DTD tiene varias limitaciones a pesar de su uso generalizado. Por ejemplo, a los elementos y atributos no se les pueden asignar tipos de forma flexible, y los elementos siempre están ordenados incluso si la aplicación no requiere que lo estén. Los esquemas XML son una nueva propuesta del W3C que proporciona una forma más potente que las DTD para describir la estructura de documentos; es un superconjunto de las DTD, de forma que permiten que los datos heredados de DTD se puedan manejar fácilmente. Un aspecto interesante es que soporta unicidad y restricciones de clave externa.

parte del interés por XML tiene sus orígenes en la creencia de que se desarrollarán más y más DTD estandarizadas. Las DTD estandarizadas permitirían un intercambio de datos fluido entre fuentes heterogéneas, un problema solucionado hoy en día mediante la implementación de protocolos especializados como **intercambio de datos electrónico** (Electronic Data Interchange, EDI) o con soluciones *ad hoc*.

Incluso en un entorno donde todos los datos XML son válidos, no es posible integrar directamente varios documentos XML haciendo corresponder elementos en sus DTD, porque aunque dos elementos tengan nombres idénticos en diferentes DTD, el significado de los elementos puede ser completamente distinto. Si ambos documentos utilizan una única DTD estándar se evita este problema. El desarrollo de DTD estandarizadas es más un proceso social que un problema de investigación, pues los participantes principales de un dominio o sector industrial dado deben cooperar.

Por ejemplo, el **lenguaje de marcado matemático** (mathematical markup language, MathML) se ha desarrollado para codificar contenido matemático en la Web. Hay dos tipos de elementos MathML. Los 28 **elementos de presentación** describen la estructura de composición de un documento; un ejemplo de ellos es el elemento `mrow`, que indica una fila horizontal de caracteres, y también el elemento `msup`, que indica una base y un subíndice. Los 75 **elementos de contenido** describen conceptos matemáticos. Un ejemplo es el elemento `plus`, que denota el operador de suma. (Un tercer tipo de elemento, el elemento `math`, se utiliza para pasar parámetros al procesador MathML.) MathML permite codificar objetos matemáticos en ambas notaciones pues los requisitos del usuario de los objetos podrían ser diferentes. Los elementos de contenido codifican el significado matemático preciso de un objeto sin ambigüedad, y la descripción puede ser utilizada por aplicaciones tales como los sistemas de álgebra computacional. Por otra parte, una buena notación puede sugerir la estructura

lógica a un humano y enfatizar aspectos clave de un objeto; los elementos de presentación permiten describir objetos matemáticos a este nivel.

Por ejemplo, considérese la siguiente ecuación sencilla:

$$x^2 - 4x - 32 = 0$$

Utilizando elementos de presentación, la ecuación se representa como sigue:

```
<mrow>
  <mrow> <msup><mi>x</mi><mn>2</mn></msup>
    <mo>-</mo>
    <mrow><mn>4</mn>
      <mo>&invisibletimes;</mo>
      <mi>x</mi>
    </mrow>
    <mo>-</mo><mn>32</mn>
  </mrow><mo>=</mo><mn>0</mn>
</mrow>
```

Utilizando elementos de contenido, la ecuación se describe como sigue:

```
<reln><eq/>
  <apply>
    <minus/>
    <apply> <power/> <ci>x</ci> <cn>2</cn> </apply>
    <apply> <times/> <cn>4</cn> <ci>x</ci> </apply>
    <cn>32</cn>
  </apply> <cn>0</cn>
</reln>
```

Obsérvese la potencia añadida que se obtiene utilizando MathML en lugar de codificar la fórmula en HTML. La forma habitual de visualizar objetos matemáticos en un objeto HTML es incluyendo imágenes que muestren los objetos, como por ejemplo en el siguiente fragmento de código:

```
<IMG SRC="images/equation.gif" ALT=" x**2 - 4x - 32 = 10 " >
```

La ecuación se codifica dentro de una etiqueta **IMG** con un formato de visualización alternativo especificado en la etiqueta **ALT**. Utilizando esta codificación de objetos matemáticos se producen los siguientes problemas de presentación. En primer lugar, la imagen normalmente se ajusta para corresponderse a un tamaño de letra determinado, y en sistemas con otros tamaños la imagen queda demasiado pequeña o demasiado grande. En segundo lugar, en sistemas con distinto color de fondo, la imagen no combina con el fondo y la resolución de la imagen es normalmente inferior cuando se imprime el documento. Además de los problemas relacionados con la diferente presentación, no es posible buscar fácilmente una fórmula o fragmento de fórmula en una página, pues no tiene una etiqueta de marcado específica.

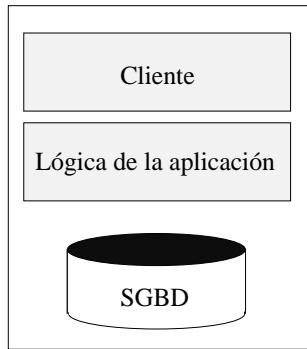


Figura 7.5 Arquitectura de una capa

7.5 LA ARQUITECTURA DE APLICACIONES DE TRES CAPAS

En este apartado se estudiará la arquitectura general de aplicaciones de Internet de procesamiento de datos. Estas aplicaciones pueden describirse en términos de tres componentes funcionales diferentes: *gestión de datos*, *lógica de la aplicación* y *presentación*. El componente que se encarga de la gestión de datos normalmente utiliza un SGBD para el almacenamiento de datos, pero la lógica de la aplicación y la presentación implican mucho más que el SGBD.

Se comenzará con una breve visión general de la historia de arquitecturas de aplicaciones de base de datos y se introducirán las arquitecturas de una capa y cliente-servidor en el Apartado 7.5.1. La arquitectura de tres capas se explicará en detalle en el Apartado 7.5.2 y se mostrarán sus ventajas en el Apartado 7.5.3.

7.5.1 Arquitecturas de una capa y cliente-servidor

En este apartado se proporciona una breve perspectiva sobre la arquitectura de tres capas comentando las arquitecturas de una capa y cliente-servidor, esta última predecesora de la primera. Inicialmente, las aplicaciones de procesamiento de datos estaban unidas en una sola capa, incluyendo el SGBD, la lógica de la aplicación y la interfaz de usuario, como se ilustra en la Figura 7.5. Estas aplicaciones normalmente se ejecutaban en un ordenador principal, y los usuarios accedían a ellas a través de *terminales simples* que lo único que podían hacer era introducir datos y mostrarlos. Este enfoque tiene la ventaja de poder ser gestionable fácilmente por un administrador central.

Las arquitecturas de una capa tienen un inconveniente importante: los usuarios esperan interfaces gráficas que requieren más potencia de cálculo que los simples terminales simples. La computación centralizada de tales interfaces requiere mucha más capacidad de procesamiento de la que tiene disponible un servidor, y por ello las arquitecturas de una capa no son adecuadas para miles de usuarios. La comercialización a gran escala de ordenadores personales y la disponibilidad de ordenadores clientes baratos llevó al desarrollo de la arquitectura de dos capas.

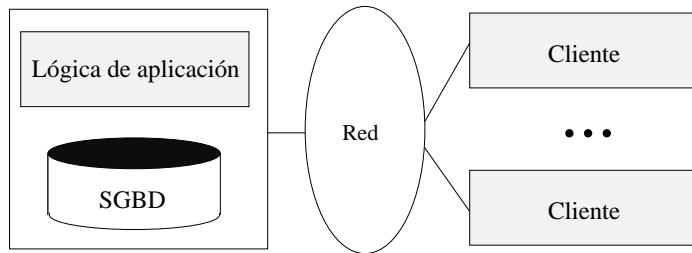


Figura 7.6 Arquitectura de dos servidores: clientes ligeros

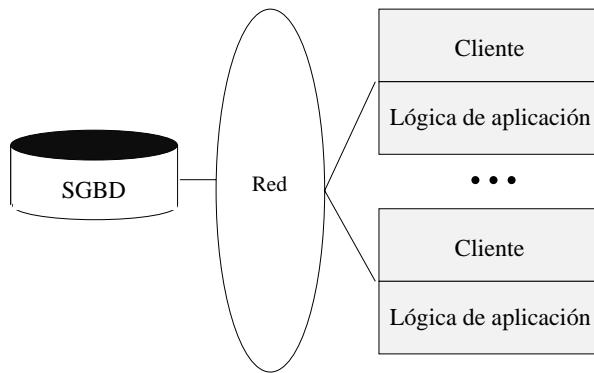


Figura 7.7 Arquitectura de dos capas: clientes pesados

Las **arquitecturas de dos capas**, también conocidas a menudo como **arquitecturas cliente-servidor** consisten en un **ordenador cliente** y un **ordenador servidor** que interactúan a través de un protocolo bien definido. La funcionalidad implementada en el cliente y en el servidor puede variar. En la **arquitectura cliente-servidor** tradicional, el cliente implementa solamente la interfaz gráfica de usuario, y el servidor implementa tanto la lógica de negocio como la gestión de datos; estos clientes suelen denominarse **clientes ligeros**, y esta arquitectura se ilustra en la Figura 7.6.

Existen otras divisiones, como clientes más potentes que implementan tanto la interfaz de usuario como la lógica de negocio, o clientes que implementan la interfaz de usuario y parte de la lógica de negocio, residiendo la otra parte en el servidor; estos clientes se denominan comúnmente **clientes pesados**. Esta arquitectura se muestra en la Figura 7.7.

En comparación con la arquitectura de una capa, las arquitecturas de dos capas separan físicamente la interfaz de usuario de la capa de gestión de datos. Para implementar arquitecturas de dos capas, ya no es posible tener terminales simples en el lado del cliente; se requieren ordenadores que ejecuten código de presentación sofisticado (y, posiblemente, la lógica de la aplicación).

En los últimos diez años se han desarrollado gran cantidad de herramientas de desarrollo cliente-servidor tales como Microsoft Visual Basic y Sybase Powerbuilder. Estas herramientas permiten el desarrollo rápido de software cliente-servidor, contribuyendo al éxito de este modelo, especialmente la versión de cliente ligero.

El modelo de cliente pesado tiene varias desventajas cuando se compara con el cliente ligero. En primer lugar, no hay un lugar centralizado donde actualizar y mantener la lógica de negocio, pues el código de aplicación se ejecuta en muchos sitios cliente. En segundo lugar, se requiere mucha confianza entre el servidor y los clientes. Por ejemplo, el SGBD de un banco tiene que confiar en (la aplicación que se ejecuta en) un cajero automático para dejar la base de datos en un estado consistente. (Una forma de resolver este problema es mediante *procedimientos almacenados*, código de aplicación fiable que se registra en el SGBD y que puede llamarse desde sentencias SQL. Los procedimientos almacenados se tratan en detalle en el Apartado 6.5.)

La tercera desventaja de la arquitectura de cliente pesado es que no es escalable respecto al número de clientes; normalmente no pueden manejar más de unos cientos de clientes. La lógica de la aplicación en el cliente emite consultas SQL al servidor y el servidor devuelve el resultado de la consulta al cliente, donde se procesa posteriormente. Podrían transferirse resultados muy grandes de consultas. (Los procedimientos almacenados pueden atenuar este cuello de botella.) En cuarto lugar, los sistemas de clientes pesados no son escalables cuando la aplicación accede a más y más bases de datos. Supóngase que hay x sistemas de bases de datos diferentes que se acceden por y clientes, entonces existen $x \cdot y$ conexiones diferentes abiertas en todo momento, lo que claramente no es una solución escalable.

Estas desventajas de los sistemas de clientes pesados y la adopción general de clientes estándares muy ligeros —particularmente, los navegadores Web— han llevado al uso generalizado de arquitecturas de clientes ligeros.

7.5.2 Arquitecturas de tres capas

La arquitectura de cliente ligero de dos capas esencialmente separa los aspectos de presentación del resto de la aplicación. La arquitectura de tres capas va un paso más allá, y también separa la lógica de la aplicación de la gestión de datos:

- **Capa de presentación.** Los usuarios requieren una interfaz natural para hacer peticiones, proporcionar datos de entrada y ver los resultados. El uso generalizado de Internet ha provocado que las interfaces basadas en Web sean cada vez más populares.
- **Capa intermedia.** La lógica de la aplicación se ejecuta aquí. Una aplicación de tipo empresarial refleja procesos de negocio complejos, y se codifica en un lenguaje de propósito general como C++ o Java.
- **Capa de gestión de datos.** Las aplicaciones Web de procesamiento de datos llevan el uso de SGBD, que son el objeto de este libro.

La Figura 7.8 muestra una arquitectura básica de tres capas. Se han desarrollado diversas tecnologías para permitir la distribución de las tres capas de una aplicación a través de múltiples plataformas hardware y diferentes emplazamientos físicos. La Figura 7.9 muestra las tecnologías relevantes de cada capa.

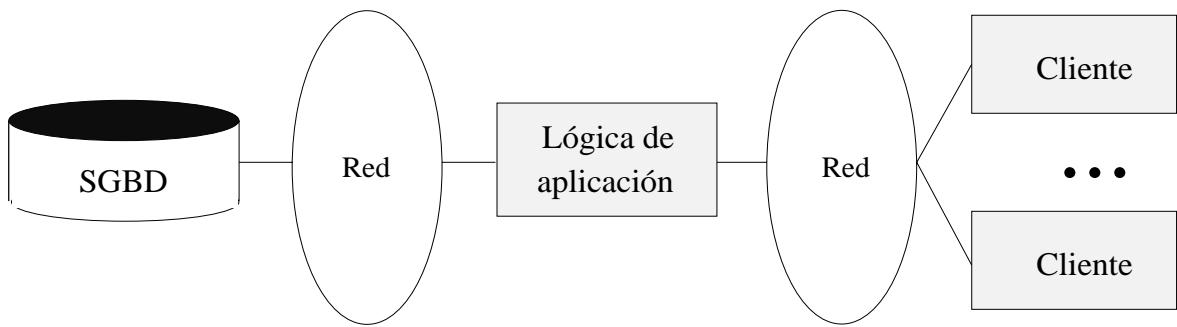


Figura 7.8 Arquitecturas de tres capas estándar

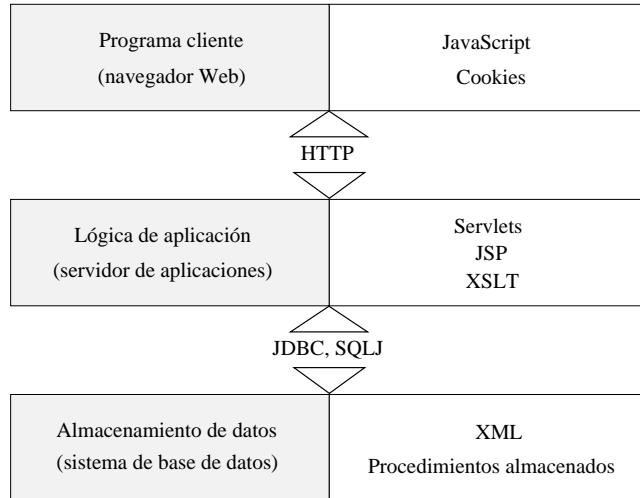


Figura 7.9 Tecnologías para las tres capas

Visión general de la capa de presentación

En el nivel de presentación es necesario proporcionar formularios para que el usuario pueda hacer solicitudes y mostrar las respuestas que genera la capa intermedia. El lenguaje de marcado de hipertexto (HTML) comentado en el Apartado 7.3 es el lenguaje básico de presentación de datos.

Es importante que esta capa de código sea fácil de adaptar a distintos dispositivos de visualización y formatos; por ejemplo, ordenadores personales frente a dispositivos de bolsillo o teléfonos móviles. Esta adaptabilidad puede alcanzarse tanto en la capa intermedia a través de distintas páginas para distintos tipos de cliente, como directamente en el cliente mediante **hojas de estilo** que especifican la forma de presentación de los datos. En este último caso, la capa intermedia es responsable de la producción de los datos apropiados en respuesta a las peticiones del usuario, mientras que la capa de presentación decide *cómo* mostrar esa información.

Las tecnologías de la capa de presentación, incluyendo las hojas de estilo, se describen en el Apartado 7.6.

Visión general de la capa intermedia

La capa intermedia ejecuta el código que implementa la lógica de negocio de la aplicación: controla los datos que son necesarios para que se pueda ejecutar cada acción, determina el flujo de control entre los pasos de una acción múltiple, controla el acceso a la capa de base de datos y normalmente compone dinámicamente páginas HTML generadas a partir de los resultados de consultas a la base de datos.

El código de la capa intermedia es responsable de respaldar todos los roles diferentes asociados a la aplicación. Por ejemplo, en una implementación de un sitio de compras en Internet sería deseable que los clientes pudieran consultar el catálogo y hacer compras, que los administradores pudieran consultar el inventario actual, y posiblemente que los analistas de datos pudieran consultar resúmenes sobre históricos de compras. Cada uno de estos roles puede requerir soporte para varias acciones complejas.

Por ejemplo, considérese un cliente que quiere comprar un artículo (después de consultar o buscar en el sitio para encontrarlo). Antes de que se pueda producir una venta, el cliente debe seguir una serie de pasos: tiene que añadir artículos a su cesta de compra, tiene que proporcionar su dirección de envío y número de tarjeta de crédito (excepto si tiene una cuenta en el sitio) y finalmente tiene que confirmar la compra con los impuestos y gastos de envío incluidos. El control del flujo entre estos pasos y la memorización de los pasos ya realizados se hace en la capa intermedia de la aplicación. Los datos utilizados durante esta serie de pasos puede implicar el acceso a bases de datos, aunque normalmente todavía no es permanente (por ejemplo, una cesta de compra no se almacena en la base de datos hasta que se confirma la venta).

La capa intermedia se tratará en detalle en el Apartado 7.7.

7.5.3 Ventajas de la arquitectura de tres capas

La arquitectura de tres capas tiene las siguientes ventajas:

- **Sistemas heterogéneos.** Las aplicaciones pueden utilizar lo mejor de diferentes plataformas y componentes software en las distintas capas. Es fácil modificar o sustituir el código de cualquier capa sin afectar a las demás capas.
- **Clientes ligeros.** Los clientes sólo necesitan disponer de suficiente capacidad de procesamiento para el nivel de presentación. Normalmente son navegadores Web.
- **Acceso integrado a datos.** En muchas aplicaciones los datos deben accederse desde varias fuentes. Esto se puede manejar de forma transparente en el nivel intermedio, donde se pueden gestionar centralizadamente las conexiones con todos los sistemas de bases de datos implicados.
- **Escalabilidad para muchos clientes.** Cada cliente es ligero y todo acceso al sistema se realiza a través de la capa intermedia. Esta capa puede compartir conexiones de bases de

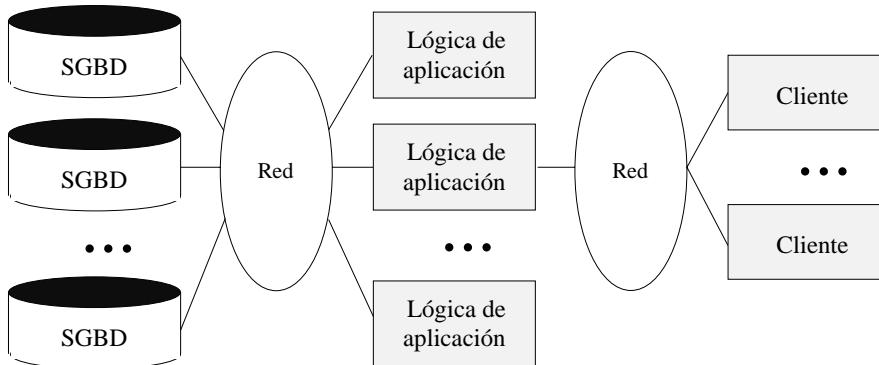


Figura 7.10 Replicación de capa intermedia y acceso a múltiples fuentes de datos

datos entre clientes, y si la capa intermedia se convierte en el cuello de botella, se pueden implantar varios servidores ejecutando el código de la capa intermedia; si la lógica se diseña apropiadamente, los clientes pueden conectarse a cualquiera de estos servidores. Este escenario se ilustra en la Figura 7.10, que también muestra el acceso de la capa intermedia a múltiples fuentes de datos. Por supuesto, depende del SGBD que cada fuente de datos sea escalable (y esto podría implicar además paralelización o réplica).

- **Beneficios del desarrollo de software.** Dividiendo la aplicación limpiamente en partes que traten la presentación, el acceso a los datos y la lógica de negocio, se consiguen muchas ventajas. La lógica de negocio está centralizada, y por tanto es fácil de mantener, depurar y modificar. La interacción entre las capas se produce a través de API bien definidas y estandarizadas. De este modo, cada capa de la aplicación puede construirse con componentes reusables que se desarrolleen, depuren y prueben individualmente.

7.6 LA CAPA DE PRESENTACIÓN

En este apartado se describen las tecnologías del lado del cliente de la arquitectura de tres capas. Se examinarán los formularios HTML como una forma especial de paso de argumentos del cliente a la capa intermedia (es decir, de la capa de presentación a la capa intermedia) en el Apartado 7.6.1. En el Apartado 7.6.2 se introducirá JavaScript, un lenguaje de secuencias de comandos que se puede utilizar para procesamiento ligero en la capa cliente (por ejemplo, para animaciones simples). Concluiremos esta descripción de tecnologías del lado del cliente presentando las hojas de estilo en el Apartado 7.6.3. Las hojas de estilo son lenguajes que permiten presentar la misma página Web con diferente formato para clientes con distintas capacidades de presentación; por ejemplo, navegadores Web frente a teléfonos móviles, o incluso el navegador Netscape frente a Microsoft Internet Explorer.

7.6.1 Formularios HTML

Los formularios HTML son la forma habitual de comunicación de datos desde la capa del cliente a la capa intermedia. El formato general de un formulario es el siguiente:

```
<FORM ACTION="página.jsp" METHOD="GET" NAME="FormInicio">
...
</FORM>
```

Un único documento HTML puede contener más de un formulario. Dentro de un formulario HTML es posible tener cualquier etiqueta HTML excepto otro elemento FORM.

La etiqueta FORM tiene tres atributos importantes:

- **ACTION**. Especifica el URI de la página a la que se envía el contenido del formulario; si no se especifica el atributo ACTION, se utiliza el URI de la página actual. En el ejemplo anterior, los datos del formulario se enviarán a la página `página.jsp`, que debe proporcionar la lógica para procesar los datos del formulario. (En el Apartado 7.7 se explicarán los métodos para leer datos de formularios en la capa intermedia.)
- **METHOD**. El método HTTP/1.0 utilizado para enviar los datos introducidos por el usuario al servidor Web. Hay dos opciones, GET y POST, cuya descripción se pospone al siguiente apartado.
- **NAME**. Este atributo permite dar un nombre al formulario. Aunque no es necesario, dar nombre a los formularios es una buena práctica. En el Apartado 7.6.2, se verá la forma de escribir programas del lado del cliente en JavaScript que se refieren a los formularios por su nombre y realizan comprobaciones sobre sus campos.

Dentro de los formularios HTML, las etiquetas INPUT, SELECT y TEXTAREA se utilizan para especificar elementos de introducción de datos del usuario; un formulario puede tener muchos elementos de cada tipo. El elemento de introducción de datos más simple es un campo INPUT, una etiqueta independiente que no tiene etiqueta de fin. Un ejemplo de etiqueta INPUT es el siguiente:

```
<INPUT TYPE="text" NAME="título">
```

La etiqueta INPUT tiene varios atributos. Los tres más importantes son TYPE, NAME y VALUE. El atributo TYPE determina el tipo del campo de entrada. Si tiene el valor `text`, el campo es de entrada de texto. Si tiene el valor `password`, es un campo de entrada de texto donde los caracteres introducidos se muestran en la pantalla como asteriscos. Si este atributo tiene el valor `reset`, es un botón sencillo que reinicia todos los campos de entrada del formulario a sus valores predeterminados. Si el atributo TYPE tiene el valor `submit`, entonces es un botón que envía los valores de los campos de entrada del formulario al servidor. Nótese que los campos de tipo `reset` y `submit` afectan a todo el formulario.

El atributo NAME de la etiqueta INPUT especifica el nombre simbólico de este campo y se utiliza para identificar el valor de este campo de entrada cuando se envía al servidor. Todas las etiquetas INPUT deben tener nombre excepto las de tipo `submit` y `reset`. En el ejemplo anterior, `título` es el valor de NAME del campo de entrada.

El atributo VALUE de una etiqueta de introducción de datos se puede utilizar para los campos de texto o de contraseña para especificar el contenido predeterminado del campo. Para los botones de envío o reinicio, VALUE determina la etiqueta del botón en la pantalla.

El formulario de la Figura 7.11 muestra dos campos de texto, un campo de texto normal y otro de contraseña. También contiene dos botones, un botón de reinicio etiquetado “Reiniciar

valores” y un botón de envío etiquetado “Iniciar sesión”. Nótese que los dos campos de entrada tienen nombre, mientras que los botones de reinicio y de envío no tienen atributo NAME.

```
<FORM ACTION="página.jsp" METHOD="GET" NAME="FormInicio">
    <INPUT TYPE="text" NAME="nombreusuario" VALUE="Juan"><P>
    <INPUT TYPE="password" NAME="clave"><P>
    <INPUT TYPE="reset" VALUE="Reiniciar valores"><P>
    <INPUT TYPE="submit" VALUE="Iniciar sesión">
</FORM>
```

Figura 7.11 Formulario HTML con dos campos de texto y dos botones

Los formularios HTML tienen otras formas de especificar entradas del usuario, tales como las etiquetas TEXTAREA y SELECT antes mencionadas, aunque aquí no se tratarán.

Paso de argumentos a secuencias de comandos del lado del servidor

Como se menciona al principio del Apartado 7.6.1, hay dos formas diferentes de enviar datos de formularios HTML al servidor Web. Si se utiliza el método GET, el contenido del formulario se reúne en un URI de consulta (como se verá a continuación) y se envía al servidor. Si se utiliza el método POST, entonces el contenido del formulario se codifica como en el método GET, pero se envían en un bloque de datos separado en lugar de agregarlos directamente al URI. De esta manera, en el método GET el contenido es directamente visible en el URI por el usuario, mientras que en el método POST el contenido se envía dentro del cuerpo del mensaje de petición y no es visible por el usuario.

El uso del método GET da a los usuarios la posibilidad de guardar en la lista de favoritos la página con el URI construido y de esta forma saltar directamente a ella en sesiones posteriores; esto no es posible con el método POST. La elección de GET frente a POST viene determinada por la aplicación y sus requisitos.

Considérese la codificación del URI cuando se utiliza el método GET. El URI tiene la siguiente forma:

acción?nombre1=valor1&nombre2=valor2&nombre3=valor3

La acción es el URI especificado en el atributo ACTION de la etiqueta FORM, o el URI del documento actual si no se especificó el atributo ACTION. Los pares “nombre=valor” son los datos introducidos por el usuario en los campos INPUT del formulario. En los campos INPUT que el usuario dejó en blanco, aparece el nombre con un valor vacío (nombre=). Como ejemplo concreto, considérese el formulario de envío de contraseña del final del apartado anterior. Suponga que el usuario introduce “Juan de la Cierva” como nombre de usuario, y “secreto” como clave. Entonces el URI de la solicitud es:

página.jsp?nombreusuario=Juan+de+la+Cierva&clave=secreto

Los datos de entrada de formularios pueden contener caracteres ASCII, como espacios en blanco, pero los URI tienen que ser una cadena de caracteres consecutivos sin espacios. Por ello, los caracteres especiales tales como espacios, “=” y otros caracteres no imprimibles se codifican de forma especial. Para crear un URI que tenga campos codificados, se realizan los siguientes tres pasos:

1. Convertir todos los caracteres especiales de nombres y valores en “%xyz”, donde “xyz” es el valor ASCII del carácter en hexadecimal. Los caracteres especiales incluyen =, &, %, + y otros caracteres no imprimibles. Obsérvese que *todos* los caracteres se podrían codificar por su valor ASCII.
2. Convertir todos los espacios en blanco en el carácter “+”.
3. Unir los nombres y valores correspondientes de una misma etiqueta INPUT de HTML con “=” y después concatenar los pares nombre-valor de las distintas etiquetas INPUT separándolos con “&” para crear un URI de la forma:
acción?nombre1=valor1&nombre2=valor2&nombre3=valor3

Nótese que para procesar los elementos de entrada del formulario HTML en la capa intermedia es necesario que el atributo ACTION de la etiqueta FORM apunte a una página, secuencia de comandos o programa que procese los valores de los campos del formulario que el usuario ha introducido. La forma de recibir valores de campos de formularios se tratan en los Apartados 7.7.1 y 7.7.3.

7.6.2 JavaScript

JavaScript es un lenguaje de secuencias de comandos en la capa cliente con el que es posible añadir programas a las páginas Web que se ejecuten directamente en el cliente (es decir, en la máquina que ejecuta el navegador Web). JavaScript se utiliza con frecuencia para los siguientes tipos de procesamiento en el cliente:

- **Detección del navegador.** JavaScript se puede utilizar para detectar el tipo de navegador y cargar una página específica para cada navegador.
- **Validación de formularios.** JavaScript permite realizar comprobaciones simples de consistencia sobre los campos del formulario. Por ejemplo, un programa JavaScript puede comprobar si un campo que pida una dirección de correo electrónico contiene el carácter ‘@’, o si el usuario ha llenado todos los campos obligatorios.
- **Control del navegador.** Esto incluye abrir páginas en otras ventanas; por ejemplo, los molestos anuncios emergentes que se ven en muchos sitios Web, que están programados usando JavaScript.

JavaScript normalmente está incorporado en un documento HTML con una etiqueta especial, SCRIPT. Esta etiqueta tiene el atributo LANGUAGE, que indica el lenguaje en el que está escrito la secuencia de comandos. Para JavaScript debe asignarse el atributo de lenguaje a JavaScript. Otro atributo de la etiqueta SCRIPT es SRC, que especifica un archivo externo con código JavaScript que se incluye automáticamente en el documento HTML. Los archivos

de código fuente JavaScript tienen habitualmente la extensión “.js”. El siguiente fragmento muestra un archivo JavaScript incluido en un documento HTML:

```
<SCRIPT LANGUAGE="JavaScript" SRC="validarForm.js"> </SCRIPT>
```

La etiqueta **SCRIPT** puede situarse dentro de comentarios HTML de manera que el código JavaScript no se muestre textualmente en los navegadores Web que no reconozcan la etiqueta **SCRIPT**. A continuación se muestra otro ejemplo de código JavaScript que crea una ventana emergente con un mensaje de bienvenida. Se encierra el código JavaScript dentro de comentarios HTML por el motivo que se acaba de mencionar.

```
<SCRIPT LANGUAGE="JavaScript">
<!--
    alert("Bienvenido a nuestra librería");
//-->
</SCRIPT>
```

JavaScript proporciona dos estilos diferentes de comentarios: los comentarios de una sola línea comienzan con los caracteres “//”, y los comentarios de varias líneas empiezan por “/*” y terminan con “*/”¹.

JavaScript tiene variables que pueden ser números, valores booleanos (cierto o falso), cadenas de caracteres y otros tipos de datos que no se verán en este libro. Las variables globales deben declararse antes de ser usadas con la palabra clave **var**, y pueden utilizarse en cualquier lugar dentro del documento HTML. Las variables locales a una función JavaScript (que se verán más adelante) no necesitan declararse. Las variables no son de un tipo fijo, sino que implícitamente tienen el tipo del dato que tienen asignado.

JavaScript dispone de los operadores de asignación habituales (=, +=, etc.), los operadores habituales aritméticos (+, -, *, /, %), de comparación (==, !=, >=, etc.) y booleanos (&& para el AND lógico, || para el OR lógico y ! para la negación). Las cadenas de caracteres se pueden concatenar utilizando el carácter “+”. El tipo de un objeto determina el comportamiento de los operadores; por ejemplo 1+1 es 2, ya que se están sumando números, mientras que “1”+“1” es “11”, pues se están concantenando cadenas de caracteres. JavaScript contiene los tipos habituales de sentencias, tales como asignaciones, sentencias condicionales (**if** (**condición**) {**sentencias**; } **else** {**sentencias**; }) y bucles (**for**, **do-while** y **while**).

JavaScript permite crear funciones mediante la palabra clave **function**: **function** **f**(**arg1**, **arg2**) {**sentencias**; }. Se puede llamar a funciones desde código JavaScript, y éstas pueden devolver valores utilizando la palabra clave **return**.

Esta introducción a JavaScript concluye con un ejemplo de una función que comprueba si los campos de usuario y contraseña de un formulario HTML no están vacíos. La Figura 7.12 muestra la función JavaScript y el formulario HTML. El código JavaScript es una función denominada **compruebaInicioVacio()** que comprueba si está vacío alguno de los dos campos de entrada del formulario llamado **FormInicio**. En esta función se utiliza la variable **formInicio** para referirse al formulario **FormInicio** mediante la variable **document** definida

¹Realmente, “<!--” también marca el inicio de un comentario de una línea, que es el motivo por el que no es necesario indicar el comentario de inicio de HTML “<!--” en el ejemplo anterior utilizando la notación de comentarios de JavaScript. Por el contrario, la marca HTML de fin de comentario “-->” debe comentarse en JavaScript, ya que de otro modo sería interpretada.

implícitamente, que representa la página HTML actual. (JavaScript incluye una biblioteca de objetos definidos implícitamente.) A continuación se comprueba si alguna de las cadenas `formInicio.usuario.value` o `formInicio.clave.value` está vacía.

La función `compruebaInicioVacio` se llama desde un manejador de eventos de formulario. Un **manejador de eventos** es una función a la que se llama si ocurre un evento sobre un objeto de una página Web. El manejador de eventos que se utiliza es `onSubmit`, que se llama si se pulsa el botón de tipo `submit` (o si el usuario pulsa Intro en un campo de texto del formulario). Si el manejador de eventos devuelve `true`, el contenido del formulario se envía al servidor, y en caso contrario no se envía.

JavaScript tiene funcionalidades que van más allá de los fundamentos explicados en este apartado; el lector interesado puede consultar las notas bibliográficas al final de este capítulo.

```
<SCRIPT LANGUAGE="JavaScript">
<!--
function compruebaInicioVacio()
{
    formInicio = document.FormInicio
    if ((formInicio.usuario.value == "") ||
        (formInicio.clave.value == ""))
    {
        alert('Por favor, introduzca usuario y clave.');
        return false;
    }
    else
        return true;
}
//-->
</SCRIPT>
<H1 ALIGN = "CENTER">Librería en Internet Benito y Norberto</H1>
<H3 ALIGN = "CENTER">Por favor introduzca su identificador de usuario y clave:</H3>
<FORM NAME = "FormInicio" METHOD="POST"
      ACTION="TablaDeContenidos.jsp"
      onSubmit="return compruebaInicioVacio()">
    Usuario: <INPUT TYPE="TEXT" NAME="usuario"><P>
    Clave: <INPUT TYPE="PASSWORD" NAME="clave"><P>
    <INPUT TYPE="SUBMIT" VALUE="Iniciar Sesión" NAME="SUBMIT">
    <INPUT TYPE="RESET" VALUE="Limpiar campos" NAME="RESET">
</FORM>
```

Figura 7.12 Validación de formulario con JavaScript

7.6.3 Hojas de estilo

Clients diferentes tienen diferentes pantallas, y por ello es necesario disponer de diferentes formas de visualizar la misma información. Por ejemplo, en el caso más simple, podríamos ser necesario utilizar diferentes tamaños de letra o colores que proporcionen alto contraste en pantallas en blanco y negro. En otro ejemplo más sofisticado, podríamos ser necesario reorganizar los objetos de la página para acomodarla a pantallas pequeñas en asistentes digitales personales (PDA). O se podría resaltarse información diferente para centrarse en alguna parte importante de la página. Una **hoja de estilo** es un método para adaptar el mismo contenido de un documento a formatos de presentación diferentes. Una hoja de estilo contiene instrucciones que indican a un navegador Web (o lo que el cliente utilice para mostrar la página Web) cómo traducir los datos de un documento en una presentación apropiada para la pantalla del cliente.

Las hojas de estilo separan el aspecto de **transformación** de la página de los aspectos de **presentación** de la página. Durante la transformación, se reorganizan los objetos del documento XML para formar una estructura diferente, para omitir partes del documento XML, para combinar dos documentos XML diferentes en uno solo. Durante la presentación se toma la estructura jerárquica del documento XML y se da formato al documento de acuerdo con el dispositivo de visualización del usuario.

El uso de hojas de estilo tiene múltiples ventajas. En primer lugar, se puede reutilizar el mismo documento muchas veces y mostrarlo de diferentes formas dependiendo del contexto. En segundo lugar, se puede adaptar la visualización a preferencias del usuario tales como tamaño de letra, estilo de color e incluso nivel de detalle. Tercero, se pueden tratar diferentes formatos de salida, como por ejemplo distintos dispositivos (portátiles o teléfonos móviles), diferentes tamaños de pantalla (DIN A4 o carta) o medios de visualización (papel o pantalla digital). En cuarto lugar, es posible estandarizar el formato de visualización dentro de una corporación y de este modo aplicar los convenios de la hoja de estilo a documentos en cualquier momento. Más aún, los cambios y mejoras que se hagan en estos convenios de visualización pueden gestionarse en un sitio central.

Hay dos lenguajes de hojas de estilo: XSL y CSS. CSS se creó para HTML con el objetivo de separar las etiquetas de formato de sus características de visualización. XSL es una extensión de CSS para documentos XML arbitrarios; además de permitir definir formas de dar formato a objetos, contiene un lenguaje de transformación que posibilita reordenar objetos. Los archivos HTML son el objetivo de CSS, mientras que para XSL son los archivos XML.

Hoja de estilo en cascada

Una **hoja de estilo en cascada** (Cascading Style Sheet, CSS) define la forma en que se muestran los elementos HTML. (En el Apartado 7.13 se introduce un lenguaje de hojas de estilo más general diseñado para documentos XML.) Los estilos normalmente están almacenados en hojas de estilo, que son archivos que contienen definiciones de estilo. Diferentes documentos HTML, tales como todos los documentos de un sitio Web, pueden referirse a la misma CSS. De este modo, es posible cambiar el formato de un sitio Web modificando un único archivo. Ésta es una forma muy apropiada de cambiar el diseño de muchas páginas Web al mismo tiempo, y un primer paso hacia la separación del contenido de la presentación.

```
BODY {BACKGROUND-COLOR: yellow}
H1 {FONT-SIZE: 36pt}
H3 {COLOR: blue}
P {MARGIN-LEFT: 50px; COLOR: red}
```

Figura 7.13 Ejemplo de hoja de estilo

En la Figura 7.13 se muestra un ejemplo de hoja de estilo. Se incluye en un archivo HTML con la siguiente línea:

```
<LINK REL="style sheet" TYPE="text/css" HREF="libros.css" />
```

Cada línea de una hoja CSS está formada por tres partes; un selector, una propiedad y un valor. Están ordenadas sintácticamente de la siguiente forma:

```
selector {propiedad: valor}
```

El **selector** es el elemento o etiqueta cuyo formato se está definiendo. La **propiedad** indica el atributo de la etiqueta cuyo valor se quiere indicar en la hoja de estilo, y el **valor** es el valor del atributo. Como ejemplo, considérese la primera línea de la hoja de estilo mostrada en la Figura 7.13:

```
BODY {BACKGROUND-COLOR: yellow}
```

Esta línea tiene el mismo efecto que cambiar el código HTML de la siguiente forma:

```
<BODY BACKGROUND-COLOR="yellow">.
```

El valor siempre debe ponerse entre comillas, pues podría estar formado por varias palabras. Se pueden indicar varias propiedades del mismo selector separándolas por punto y coma, como se muestra en la última línea del ejemplo de la Figura 7.13:

```
P {MARGIN-LEFT: 50px; COLOR: red}
```

Las hojas de estilo en cascada tienen una sintaxis extensa; las notas bibliográficas al final del capítulo incluyen libros y recursos en línea sobre CSS.

XSL

XSL es un lenguaje para expresar hojas de estilo. Una hoja de estilo XSL es, como CSS, un archivo que describe cómo mostrar un documento XML de un tipo determinado. XSL comparte la funcionalidad de CSS y es compatible con él (aunque utiliza una sintaxis diferente).

Las capacidades de XSL exceden enormemente la funcionalidad de CSS. XSL incluye el lenguaje de **transformación XSL**, o XSLT, un lenguaje que permite transformar un documento XML en otro documento XML con otra estructura. Por ejemplo, con XSLT es posible cambiar el orden de los elementos que se están mostrando (por ejemplo, ordenándolos), procesar elementos más de una vez, suprimir elementos en un lugar y presentarlos en otro, y añadir texto generado para la presentación.

XSL también contiene el lenguaje **Path XML** (XML Path Language, XPath), un lenguaje que permite referirse a partes de un documento XML. Veremos XPath en el Capítulo 18. XSL también incluye el objeto de formato XSL (XSL Formatting Object), una forma de dar formato al resultado de una transformación XSL.

7.7 LA CAPA INTERMEDIA

En este apartado se estudian las tecnologías de la capa intermedia. Las aplicaciones para la capa intermedia de primera generación eran programas autónomos escritos en un lenguaje de programación de propósito general como C, C++ o Perl. Los programadores se dieron cuenta rápidamente que la interacción con una aplicación autónoma era bastante costosa; esta sobrecarga incluye el inicio de la aplicación cada vez que es invocada y el cambio de procesos entre el servidor Web y la aplicación. Por ello, estas interacciones no se dimensionan bien con un gran número de usuarios concurrentes. Esto llevó al desarrollo de los **servidores de aplicaciones**, que proporcionan el entorno de ejecución de diversas tecnologías que se pueden usar para programar componentes de aplicaciones de la capa intermedia. La mayor parte de los sitios Web a gran escala utilizan un servidor de aplicaciones para ejecutar código de aplicación en la capa intermedia.

La descripción en este libro de las tecnologías de la capa intermedia refleja esta evolución. Se comenzará en el Apartado 7.7.1 con la interfaz de pasarela común, una interfaz que se utiliza para transmitir argumentos desde formularios HTML a programas de aplicación que se ejecutan en la capa intermedia. Los servidores de aplicaciones se introducen en el Apartado 7.7.2. Después se describen las tecnologías para escribir la lógica de las aplicaciones en la capa intermedia: servlets de Java (Apartado 7.7.3) y páginas de servidor de Java (Apartado 7.7.4). Otra funcionalidad importante es el mantenimiento del estado en el componente de la capa intermedia de la aplicación según el componente cliente atraviesa una serie de pasos para completar una transacción (por ejemplo, la compra de una cesta de artículos o la reserva de un vuelo). En el Apartado 7.7.5 se tratan las cookies, una aproximación al mantenimiento del estado.

7.7.1 CGI: la interfaz de pasarela común

La interfaz de pasarela común (Common Gateway Interface, CGI) conecta formularios HTML con programas de aplicación. Es un protocolo que define el modo en que se pasan los argumentos de los formularios a los programas en el lado del servidor. No se entrará en los detalles del protocolo CGI real, pues hay bibliotecas que permiten a los programas de aplicación obtener los argumentos del formulario HTML; en breve se presentará un ejemplo en un programa CGI. Los programas que se comunican con el servidor Web a través de CGI se denominan a menudo **secuencias de comandos CGI**, pues muchos programas de aplicación fueron escritos en lenguajes de secuencias de comandos como Perl.

Como ejemplo de programa que interactúa con un formulario HTML mediante CGI, considérese la página de ejemplo mostrada en la Figura 7.14. Esta página Web contiene un formulario donde un usuario puede escribir el nombre de un autor. Si el usuario pulsa el botón “Enviar”, la secuencia de comandos en Perl “buscaLibros.cgi” mostrada en la Figura

```

<HTML><HEAD><TITLE>Base de datos de la librería</TITLE></HEAD>
<BODY>
<FORM ACTION="buscaLibros.cgi" METHOD=POST>
    Escriba un nombre de autor:
    <INPUT TYPE="text" NAME="nombreAutor"
           SIZE=30 MAXLENGTH=50>
    <INPUT TYPE="submit" value="Enviar">
    <INPUT TYPE="reset" VALUE="Limpiar formulario">
</FORM>
</BODY></HTML>

```

Figura 7.14 Página Web de ejemplo en la que se envía la entrada de un formulario a una secuencia de comandos CGI

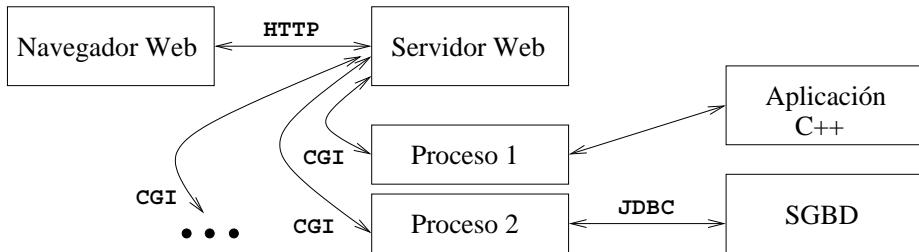


Figura 7.15 Estructura de procesos con secuencias de comandos CGI

7.14 se ejecuta como un proceso separado. El protocolo CGI define la manera en que se realiza la comunicación entre el formulario y la secuencia de comandos. La Figura 7.15 ilustra los procesos creados cuando se utiliza el protocolo CGI.

La Figura 7.16 muestra la secuencia de comandos CGI, escrita en Perl. Para simplificar se omite el código de comprobación de errores. Perl es un lenguaje interpretado que se utiliza frecuentemente para programación CGI y existen muchas bibliotecas Perl, llamadas **módulos**, que proporcionan interfaces de alto nivel con el protocolo CGI. En este ejemplo se usa una de ellas, llamada **biblioteca DBI**. El módulo CGI es una práctica colección de funciones para crear secuencias de comandos CGI. En la parte 1 de la secuencia de comandos de ejemplo se extrae el argumento del formulario HTML que se pasa del cliente:

```
$nombreAutor = $datosEntrada->param('nombreAutor');
```

Obsérvese que el nombre del parámetro **nombreAutor** se utilizó en el formulario de la Figura 7.14 para nombrar el primer campo de entrada. El protocolo CGI abstrae la implementación real de cómo se devuelve la página Web al navegador; esta página consiste simplemente en la salida del programa, y se empieza a componer en la parte 2. Todo lo que la secuencia de comandos escribe en sentencias **print** es parte de la página Web construida dinámicamente que se devuelve al navegador. La parte 3 termina concatenando las etiquetas de fin de formato de la página resultante.

```

#!/usr/bin/perl
use CGI;

### parte 1
$datosEntrada = new CGI;
$datosEntrada->header();
$nombreAutor = $datosEntrada->param('nombreAutor');

### parte 2
print("<HTML><TITLE>Prueba de paso de argumentos</TITLE>");
print("El usuario pasó el siguiente argumento:");
print("nombreAutor: ", $nombreAutor);

### parte 3
print ("</HTML>");
exit;

```

Figura 7.16 Secuencia de comandos simple en Perl

7.7.2 Servidores de aplicaciones

La lógica de la aplicación puede hacerse cumplir por medio de programas del lado del servidor invocados según el protocolo CGI. No obstante, como cada solicitud de página produce la creación de un proceso nuevo, esta solución no se dimensiona bien con un gran número de solicitudes simultáneas. Este problema de rendimiento llevó al desarrollo de programas especializados denominados **servidores de aplicaciones**. Un servidor de aplicaciones mantiene una memoria intermedia de hilos o procesos y los utiliza para ejecutar solicitudes. De este modo, evita el coste de creación de un proceso nuevo por cada solicitud.

Los servidores de aplicaciones han evolucionado a paquetes flexibles de capa intermedia que proporcionan muchas funciones además de eliminar la sobrecarga por la creación de procesos. Facilitan los accesos concurrentes a diversas fuentes de datos heterogéneas (por ejemplo, incluyendo controladores JDBC) y proporcionan servicios de **gestión de sesiones**. Los procesos de negocio implican normalmente varios pasos. Los usuarios esperan que el sistema mantenga la continuidad durante una sesión multipaso. Se pueden utilizar diversos métodos para identificar una sesión, tales como **cookies**, extensiones de URI y campos ocultos en formularios HTML. Los servidores de aplicaciones proporcionan la funcionalidad para detectar el momento en que se inicia y termina una sesión, y para mantener las sesiones de cada usuario. También ayudan a mantener el acceso seguro a bases de datos, incluyendo un mecanismo general de identificación de usuarios. (Para más información sobre seguridad, véase el Capítulo 14.)

En la Figura 7.17 se muestra una posible arquitectura de sitio Web con un servidor de aplicaciones. El cliente (un navegador Web) interactúa con el servidor Web por medio del protocolo HTTP. El servidor Web entrega páginas estáticas HTML o XML directamente al

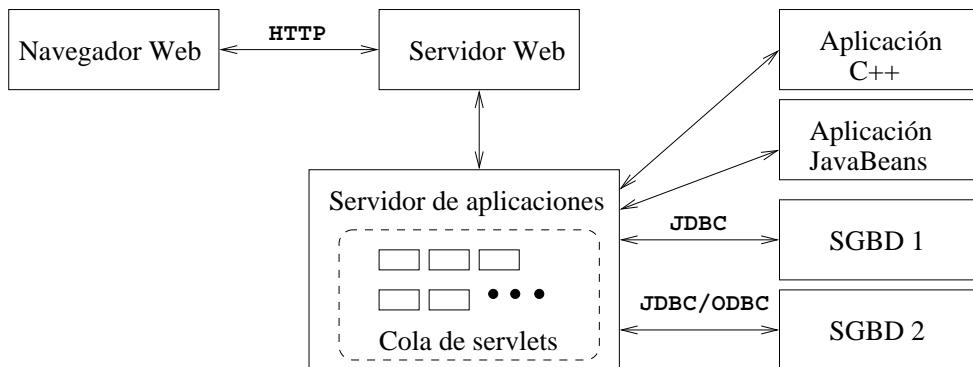


Figura 7.17 Estructura de procesos en la arquitectura de servidor de aplicaciones

cliente. Para componer páginas dinámicas, el servidor Web envía una solicitud al servidor de aplicaciones. Éste se pone en contacto con una o más fuentes de datos para recuperar la información necesaria o envía solicitudes de actualización a las fuentes de datos. Después de finalizar la interacción con las fuentes de datos, el servidor de aplicaciones compone la página Web e informa del resultado al servidor Web, que recupera la página y la entrega al cliente.

La ejecución de la lógica de negocio en el servidor Web, denominada **procesamiento del lado del servidor**, se ha convertido en un modelo estándar para la implementación de procesos de negocio más complejos sobre Internet. Hay muchas tecnologías de procesamiento en el servidor y sólo se mencionan algunas en este apartado; el lector interesado puede remitirse a las notas bibliográficas al final del capítulo.

7.7.3 Servlets

Los **servlets de Java** son fragmentos de código Java que se ejecutan en la capa intermedia, tanto en servidores Web como en servidores de aplicaciones. Existen convenios especiales sobre el modo de leer los datos de entrada del usuario y de escribir el resultado generado por un servlet. Son realmente independientes de la plataforma y por ello se han hecho muy populares entre los desarrolladores Web.

Como los servlets son programas Java, son muy versátiles. Por ejemplo, pueden construir páginas Web, acceder a bases de datos y mantener el estado. Tienen acceso a todas las API de Java, incluyendo JDBC. Todos los servlets deben implementar la interfaz **Servlet**. En la mayor parte de los casos, los servlets extienden la clase **HttpServlet** para que los servidores puedan comunicarse con los clientes mediante HTTP. Esta clase proporciona métodos tales como **doGet** y **doPost** para recibir argumentos de formularios HTML y enviar su resultado al cliente por HTTP. Los servlets que se comuniquen usando otros protocolos (como ftp) deben extender la clase **GenericServlet**.

Los servlets son clases compiladas de Java que se ejecutan y mantienen en un **contenedor de servlets**. Este contenedor gestiona el tiempo de vida de cada servlet creándolos y destruyéndolos. Aunque los servlets pueden responder a cualquier tipo de solicitud, se usan

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PlantillaServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        // Uso de 'out' para enviar contenido al navegador
        out.println("Hola a todos");
    }
}

```

Figura 7.18 Plantilla de Servlet

normalmente para extender aplicaciones alojadas en servidores Web. Para estas aplicaciones, existe una biblioteca de clases servlet específicas para HTTP.

Los servlets normalmente manejan solicitudes de formularios HTML y mantienen el estado entre el cliente y el servidor. En el Apartado 7.7.5 se trata cómo se mantiene el estado. Una plantilla de una estructura de servlet genérica se muestra en la Figura 7.18. Este servlet sólo escribe las palabras “Hola a todos”, pero muestra la estructura general de un servlet completo. El objeto **request** se utiliza para leer los datos de un formulario HTML. El objeto **response** permite especificar el código de estado de la respuesta HTTP y las cabeceras de dicha respuesta. El objeto **out** se utiliza para componer el contenido que se devuelve al cliente.

Recuérdese que HTTP devuelve la línea de estado, una cabecera, una línea en blanco y después el contexto. Ahora mismo el servlet de ejemplo solo devuelve texto plano. Se puede extender este servlet seleccionando el tipo de contenido HTML de la siguiente forma:

```

PrintWriter out = response.getWriter();
String tipoDoc =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
     \"Transitional//EN\"> \n";
out.println(tipoDoc +
    "<HTML>\n" +
    "<HEAD><TITLE>Hola WWW</TITLE></HEAD>\n" +
    "<BODY>\n" +
    "<H1>Hola WWW</H1>\n" +
    "</BODY></HTML>");
```

¿Qué ocurre durante la vida de un servlet? Se llama a diversos métodos en diferentes fases del desarrollo de un servlet. Cuando una página solicitada es un servlet, el servidor Web remite la petición al contenedor de servlets, que crea un ejemplar del servlet si fuera

necesario. En el momento de la creación del servlet, el contenedor llama al método `init()`, y antes de eliminar el servlet del contenedor, llama a su método `destroy()`.

Cuando un contenedor de servlets llama a un servlet porque se ha solicitado una página, comienza con el método `service()`, que llama por defecto a uno de los siguientes métodos basados en el método de transferencia HTTP: `doGet()` para una solicitud GET de HTTP, o `doPost()` para una solicitud POST. Este reenvío automático permite al servidor realizar diferentes tareas dependiendo del método de transferencia HTTP. Normalmente no se sobrescribe el método `service()`, a no ser que se quiera programar un servlet que maneje las solicitudes POST y GET de forma idéntica.

Esta descripción de los servlets finaliza con un ejemplo, mostrado en la Figura 7.19, que ilustra cómo pasar argumentos de un formulario HTML a un servlet.

7.7.4 Páginas de servidor de Java

En el apartado anterior se ha visto el modo de usar programas Java en la capa intermedia para codificar la lógica de la aplicación y generar páginas Web dinámicamente. Si fuese necesario generar una salida HTML, se tendría que escribir en el objeto `out`. De este modo, se puede pensar que los servlets son código Java que expresa la lógica de la aplicación, con HTML incorporado para generar el resultado.

Las páginas de servidor de Java (JavaServer pages, JSP) intercambian los roles de salida y lógica de aplicación. Están escritas en HTML con código de tipo servlet incorporado en etiquetas especiales HTML. De esta manera, comparadas con los servlets, las páginas de servidor de Java son más convenientes para desarrollar rápidamente interfaces que contengan partes sencillas de la lógica de la aplicación, mientras que los servlets vienen mejor para una lógica más compleja.

Aunque hay una gran diferencia para el programador, la capa intermedia maneja las páginas de servidor de Java de una forma muy sencilla: normalmente se compilan a un servlet, el cual es gestionado por un contenedor de servlets de forma análoga a otros servlets.

El fragmento de código de la Figura 7.20 muestra un ejemplo de JSP sencillo. Hacia la mitad del código HTML se accede a la información que se pasa desde un formulario.

7.7.5 Mantenimiento del estado

Como se ha comentado en los apartados anteriores, es necesario mantener el **estado** de un usuario entre diferentes páginas. Como ejemplo, considérese un usuario que quiere realizar una compra en el sitio Web Benito y Norberto. El usuario primero debe añadir artículos en su cesta de compra, que persiste mientras navega en el sitio Web. De esta manera, se usa la noción de estado principalmente para mantener información mientras el usuario navega por un sitio Web.

HTTP es un protocolo sin estado. Se dice que una interacción con un servidor Web es **sin estado** si no se retiene ninguna información entre una petición y la siguiente. Se dice que la interacción **tiene estado** si se almacena algo en la memoria entre peticiones al servidor, y se toman diferentes acciones dependiendo del contenido almacenado.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class LeerNombreUsuario extends HttpServlet {
    public void doGet(HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<BODY>\n" +
                   "<H1 ALIGN=CENTER> Nombre de usuario: </H1>\n" +
                   "<UL>\n" +
                   " <LI>titulo: " +
                   + request.getParameter("idusuario") + "\n" +
                   + request.getParameter("clave") + "\n" +
                   "</UL>\n" +
                   "</BODY></HTML>"); }
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response); }
}
```

Figura 7.19 Extracción del nombre de usuario y la clave de un formulario

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
                         Transitional//EN">
<HTML>
<HEAD><TITLE>Bienvenido a Benito y Norberto</TITLE></HEAD>
<BODY>
    <H1>Bienvenido</H1>
    <% String nombre="NewUser";
        if (request.getParameter("username") != null) {
            nombre=request.getParameter("username");
        }
    %>
    Ha iniciado sesión como el usuario <%=nombre%>
    <P>
        HTML normal para el resto de página de la tienda en línea.
</BODY>
</HTML>

```

Figura 7.20 Lectura de parámetros del formulario en JSP

En el ejemplo de Benito y Norberto es necesario mantener la cesta de compra de un usuario. Como el estado no está encapsulado en el protocolo HTTP, tiene que mantenerse bien en el servidor o bien en el cliente. Como el protocolo no tiene estado a propósito, se discutirán a continuación las ventajas y desventajas de esta decisión de diseño. En primer lugar, un protocolo sin estado es fácil de programar y de usar, y está muy bien para aplicaciones que sólo requieren recuperar información estática. Además, no se utiliza memoria extra para mantener el estado, y de esta forma el protocolo en sí es muy eficiente. Por otro lado, sin algún mecanismo adicional en las capas de presentación e intermedia, no hay registro de peticiones anteriores, y no es posible programar cestas de compra o inicios de sesión.

Como no es posible mantener el estado en el protocolo HTTP, ¿dónde se podría mantener? Existen básicamente dos posibilidades. Se puede mantener el estado en la capa intermedia almacenando información en la memoria local de la lógica de la aplicación, o incluso en el sistema de base de datos. O bien se puede mantener el estado en el lado del cliente almacenando los datos en forma de *cookies*. En los siguientes apartados se examinarán estas dos formas de mantener el estado.

Mantenimiento del estado en la capa intermedia

En la capa intermedia existen varias posibilidades sobre *dónde* mantener el estado. En primer lugar, se podría mantener en la última capa, el servidor de bases de datos. El estado sobreviviría a caídas del sistema, pero se necesita un acceso a la base de datos para consultar y actualizar el estado, un potencial cuello de botella de rendimiento. Una alternativa es almacenar el estado en la memoria principal de la capa intermedia. Pero esto tiene el inconveniente

de que esta información es volátil y podría ocupar gran cantidad de memoria. También es posible almacenar el estado en archivos locales de la capa intermedia, como un compromiso entre las alternativas anteriores.

Una regla general es utilizar mantenimiento del estado en la capa intermedia o en la base de datos sólo para aquellos datos que necesitan permanecer en muchas sesiones de usuario diferentes. Algunos ejemplos de estos datos son los pedidos anteriores del cliente, la grabación del recorrido que hace un usuario por el sitio Web, u otras elecciones permanentes que el usuario realiza, tales como las decisiones sobre la distribución personalizada del sitio Web, los tipos de mensajes que el usuario desea recibir, etc. Como estos ejemplos ilustran, la información de estado se centra habitualmente en los usuarios que interactúan con el sitio Web.

Mantenimiento del estado en la capa de presentación: cookies

Otra posibilidad es almacenar el estado en la capa de presentación y pasarlo a la capa intermedia con cada solicitud HTTP. La falta de estado del protocolo HTTP se resuelve fundamentalmente enviando información adicional con cada solicitud. Esta información se denomina cookie.

Una **cookie** es una colección de pares *(nombre, valor)* que puede manipularse en las capas de presentación e intermedia. Las cookies son fáciles de usar en servlets de Java y en páginas de servidor de Java, y proporcionan una forma sencilla de hacer persistentes en el cliente datos no esenciales. Sobreviven a varias sesiones del cliente porque persisten en la caché del navegador incluso después de que se cierre.

Una desventaja de las cookies es que a menudo se perciben como invasivas, y muchos usuarios deshabilitan las cookies en sus navegadores, ya que estos programas permiten a los usuarios evitar que se salven cookies en sus máquinas. Otra desventaja es que los datos de una cookie están limitados actualmente a 4Kb, aunque para la mayor parte de las aplicaciones esto no es un problema.

Se pueden utilizar cookies para almacenar información tal como la cesta de compra del usuario, información de inicio de sesión y otras elecciones no permanentes hechas en la sesión actual.

A continuación se estudiará la gestión de cookies desde servlets en la capa intermedia.

El API de cookies para servlets

Una cookie se guarda en el cliente en un pequeño archivo de texto que contiene pares *(nombre, valor)*, donde tanto el nombre como el valor son cadenas de caracteres. Las cookies nuevas se crean mediante la clase Java **Cookie** en el código de aplicación de la capa intermedia:

```
Cookie cookie = new Cookie("nombreusuario","invitado");
cookie.setDomain("www.tiendalibros.com");
cookie.setSecure(false);           // no se requiere SSL
cookie.setMaxAge(60*60*24*7*31); // tiempo de vida de un mes
response.addCookie(cookie);
```

Se examinará cada parte de este código. En primer lugar, se crea un objeto Cookie nuevo con el par *<nombre, valor>* especificado. Después se asignan valores a los atributos de la cookie; algunos de los más comunes se muestran a continuación:

- **setDomain** y **getDomain**: el dominio especifica el sitio Web que va a recibir la cookie. El valor predeterminado de este atributo es el dominio que creó la cookie.
- **setSecure** y **getSecure**: si este atributo es **true**, la cookie se envía sólo si se está usando una versión segura del protocolo HTTP, como SSL.
- **setMaxAge** y **getMaxAge**: el atributo **MaxAge** determina el tiempo de vida de la cookie en segundos. Si el valor de **MaxAge** es menor o igual a cero, la cookie se borra cuando se cierra el navegador.
- **setName** y **getName**: estas funciones no se han usado el fragmento de código de ejemplo; permiten dar un nombre a la cookie.
- **setValue** y **getValue**: estas funciones permiten asignar y leer el valor de la cookie.

La cookie se añade al objeto **response** del servlet que se envía al cliente. Una vez que se recibe una cookie de un sitio (www.tiendalibros.com en este ejemplo), el navegador del cliente la agrega a todas las solicitudes HTTP que envía a este sitio, hasta que la cookie expira.

Se puede acceder a los contenidos de una cookie en el código de la capa intermedia mediante el método **getCookies()** del objeto **request**, que devuelve un vector de objetos Cookie. El siguiente fragmento de código lee el vector y busca la cookie con nombre “**nombreusuario**”.

```
Cookie[] cookies = request.getCookies();
String elUsuario;
for(int i=0; i < cookies.length; i++) {
    Cookie cookie = cookies[i];
    if (cookie.getName().equals("nombreusuario"))
        elUsuario = cookie.getValue();
}
```

Se puede hacer una prueba sencilla para determinar si el usuario ha deshabilitado las cookies: enviar una cookie al usuario, y después comprobar que el objeto **request** devuelto contiene la cookie. Nótese que una cookie nunca debe contener una contraseña u otros datos privados sin cifrar, pues el usuario puede fácilmente inspeccionar, modificar y borrar cualquier cookie en cualquier momento, incluso en medio de una sesión. La lógica de la aplicación debe tener suficientes comprobaciones de consistencia para asegurar que los datos de la cookie son válidos.

7.8 CASO DE ESTUDIO: LA LIBRERÍA EN INTERNET

TiposBD ahora pasan a la implementación de la capa de aplicación y consideran alternativas para conectar el SGBD a la Web.

Empiezan considerando la gestión de la sesión. Por ejemplo, los usuarios que inician sesión en el sitio no quieren volver a introducir sus números de identificación de cliente cuando consultan el catálogo y seleccionan los libros a comprar. La gestión de la sesión tiene que extenderse a todo el proceso de seleccionar libros, añadirlos a la cesta de compra (posiblemente eliminando libros de la cesta) y de pasar por caja y pagar los libros.

TiposBD entonces consideran si las páginas Web de libros deberían ser estáticas o dinámicas. Si hay una página Web estática por cada libro, entonces es necesario un campo extra de la base de datos en la relación Libros que contenga la ubicación del archivo. Incluso si se habilitan diseños de página especiales para los libros, es una solución muy laboriosa. TiposBD convencen a B&N para componer dinámicamente la página Web de un libro a partir de una plantilla estándar instanciada con información sobre el libro en la relación Libros. De esta forma, TiposBD no usan páginas HTML estáticas, como la mostrada en la Figura 7.1, para mostrar el inventario.

TiposBD contemplan el uso de XML como formato de intercambio de datos entre el servidor de base de datos y la capa intermedia, o la capa intermedia y el cliente. La representación de los datos en XML en la capa intermedia mostrada en las Figuras 7.2 y 7.3 permitiría una integración más fácil de otras fuentes de datos en el futuro, pero B&N no prevén necesitar esta integración, por lo que TiposBD deciden no usar intercambio de datos XML en este momento.

TiposBD diseñan la lógica de la aplicación de la siguiente forma. Piensan que habrá cinco páginas Web diferentes:

- **index.jsp**: la página de inicio de Benito y Norberto. Éste es el punto de entrada principal a la tienda. Esta página tiene campos de texto de búsqueda y botones que permiten al usuario buscar por nombre de autor, ISBN o título del libro. También hay un enlace a la página que muestra la cesta de compra, **cesta.jsp**.
- **inicio.jsp**: permite iniciar sesión a los usuarios registrados. En este caso TiposBD utilizan un formulario HTML similar al mostrado en la Figura 7.11. En la capa intermedia usan un fragmento de código similar al mostrado en la Figura 7.19 y páginas de servidor de Java como la mostrada en la Figura 7.20.
- **busqueda.jsp**: lista todos los libros de la base de datos que satisfacen la condición de búsqueda especificada por el usuario. El usuario puede añadir artículos de la lista a la cesta de compra; cada libro tiene un botón a su lado para hacerlo. (Si el artículo ya está en la cesta de compra, incrementa la cantidad en uno.) También hay un contador que muestra el número total de artículos que actualmente hay en la cesta de compra. (TiposBD toman nota de que una cantidad de cinco unidades del mismo artículo en la cesta debe indicar una compra total de cinco.) La página **busqueda.jsp** también contiene un botón que dirige al usuario a **cesta.jsp**.
- **cesta.jsp**: lista todos los libros que actualmente están en la cesta de compra. El listado debe incluir todos los artículos con nombre de producto, precio, un cuadro de texto para la cantidad (que el usuario puede utilizar para cambiar cantidades de artículos) y un botón para eliminar el artículo de la cesta. Esta página tiene otros tres botones: uno para continuar comprando (que devuelve al usuario a la página **index.jsp**), un segundo botón

para actualizar la cesta de compra con las cantidades modificadas de los cuadros de texto, y un tercer botón para hacer el pedido, que dirige al usuario a la página `confirmar.jsp`.

- `confirmar.jsp`: lista el pedido completo hasta ahora y permite al usuario introducir su información de contacto o identificador de cliente. Hay dos botones en esta página: el primero para cancelar el pedido, y el segundo para enviar el pedido final. El botón de cancelación vacía la cesta de compra y devuelve al usuario a la página de inicio. El botón de envío actualiza la base de datos con el nuevo pedido, vacía la cesta de compra, y devuelve al usuario a la página de inicio.

TiposBD también contemplan el uso de JavaScript en la capa de presentación para comprobar los datos introducidos por el usuario antes de enviarlos a la capa intermedia. Por ejemplo, en la página `inicio.jsp`, TiposBD piensan escribir código JavaScript similar al mostrado en la Figura 7.12.

Esto deja a TiposBD una decisión final: cómo conectar las aplicaciones al SGBD. Consideran las dos alternativas principales presentadas en el Apartado 7.7: secuencias de comandos CGI frente al uso de una infraestructura de servidor de aplicaciones. Si utilizan secuencias de comandos CGI, tendrían que codificar la lógica de gestión de sesión —no es una tarea fácil—. Si utilizan un servidor de aplicaciones, pueden hacer uso de toda la funcionalidad que éste proporciona. Por tanto, recomiendan que B&N implemente el procesamiento en el lado del servidor utilizando un servidor de aplicaciones.

B&N acepta la decisión de usar un servidor de aplicaciones, pero decide que el código no debe ser específico de ninguno en particular, pues no quieren atarse a un fabricante. TiposBD están de acuerdo y proceden a construir las siguientes partes:

- TiposBD diseñan páginas de alto nivel que permiten a los clientes navegar por el sitio Web, así como diversos formularios de búsqueda y presentación de resultados.
- Suponiendo que TiposBD seleccionan un servidor de aplicaciones basado en Java, tienen que escribir servlets de Java para procesar las peticiones de los formularios. Potencialmente, podrían reusar JavaBeans existentes (posiblemente disponibles comercialmente). Pueden utilizar JDBC como interfaz de base de datos (en el Apartado 6.2 se pueden encontrar ejemplos de código JDBC). En lugar de programar servlets, pueden recurrir a páginas de servidor de Java y anotar las páginas con etiquetas de marcado JSP.
- TiposBD seleccionan un servidor de aplicaciones que utiliza etiquetas de marcado propietarias, pero debido a su acuerdo con B&N no les está permitido utilizar esas etiquetas en su código.

Obsérvese, por completitud, que si TiposBD y B&N hubieran acordado utilizar secuencias de comandos CGI, TiposBD tendrían que hacer las siguientes tareas:

- Crear las páginas de alto nivel que permitan a los usuarios navegar por el sitio y diversos formularios para búsquedas por ISBN, nombre de autor o título. En la Figura 7.1 se muestra una página de ejemplo con un formulario de búsqueda. Además de los formularios de entrada, TiposBD deben desarrollar las presentaciones apropiadas para los resultados.
- Desarrollar la lógica para seguir la pista a una sesión de cliente. La información relevante debe almacenarse bien en el lado del servidor o bien en el navegador del cliente mediante cookies.

- Escribir las secuencias de comandos que procesan las solicitudes del usuario. Por ejemplo, un cliente puede utilizar un formulario llamado “Buscar libros por título” para introducir un título y buscar libros con ese título. La interfaz CGI se comunica con una secuencia de comandos que procesa la solicitud. Un ejemplo de esta secuencia de comandos escrita en Perl y que utiliza la librería DBI para acceso a datos se muestra en la Figura 7.16.

Hasta aquí se ha cubierto solamente la interfaz del cliente, es decir, la parte del sitio Web que está expuesta a los clientes de B&N. TiposBD también tienen que agregar aplicaciones que permitan a los empleados y al propietario de la tienda consultar y acceder a la base de datos y a generar informes de las actividades del negocio.

En la página Web de este libro se pueden encontrar los archivos completos de este caso de estudio.

7.9 PREGUNTAS DE REPASO

Las respuestas a las preguntas de repaso se pueden encontrar en los apartados indicados.

- ¿Qué son los URI y los URL? (**Apartado 7.2.1**)
- ¿Cómo funciona el protocolo HTTP? ¿Qué es un protocolo sin estado? (**Apartado 7.2.2**)
- Explíquense los conceptos fundamentales de HTML. ¿Por qué se utiliza solamente para presentación de datos y no para intercambio de datos? (**Apartado 7.3**)
- ¿Cuáles son los defectos de HTML y cómo los resuelve XML? (**Apartado 7.4**)
- ¿Cuáles son los componentes principales de un documento XML? (**Apartado 7.4.1**)
- ¿Por qué hay DTD en XML? ¿Qué es un documento XML bien formado? ¿Qué es un documento XML válido? Dese un ejemplo de un documento XML que sea válido pero no bien formado, y viceversa. (**Apartado 7.4.2**)
- ¿Cuál es el rol de los DTD específicos de dominio? (**Apartado 7.4.3**)
- ¿Qué es una arquitectura de tres capas? ¿Qué ventajas ofrece respecto a las arquitecturas de una y dos capas? Dese una breve visión general de la funcionalidad de cada una de las tres capas. (**Apartado 7.5**)
- Explíquese cómo las arquitecturas de tres capas resuelven cada uno de los siguientes problemas de las aplicaciones de base de datos en Internet: heterogeneidad, clientes ligeros, integración de datos, escalabilidad, desarrollo de software. (**Apartado 7.5.3**)
- Escríbase un formulario HTML. Describanse todos los componentes de un formulario HTML. (**Apartado 7.6.1**)
- ¿Cuál es la diferencia entre los métodos HTML GET y POST? ¿Cómo funciona la codificación URI de un formulario HTML? (**Apartado 7.11**)
- ¿Para qué se utiliza JavaScript? Escríbase una función JavaScript que compruebe si un elemento de un formulario HTML contiene una dirección de correo electrónico sintácticamente válida. (**Apartado 7.6.2**)

- ¿Qué problema resuelven las hojas de estilo? ¿Cuáles son las ventajas de utilizar hojas de estilo? (**Apartado 7.6.3**)
- ¿Qué son las hojas de estilo en cascada? Explíquense los componentes de las hojas de estilo en cascada. ¿Qué es XSL y qué diferencias tiene respecto a CSS? (**Apartados 7.6.3 y 7.13**)
- ¿Qué es CGI y qué problema resuelve? (**Apartado 7.7.1**)
- ¿Qué son los servidores de aplicaciones y qué diferencias tienen respecto a los servidores Web? (**Apartado 7.7.2**)
- ¿Qué son los servlets? ¿Cómo los servlets manejan datos de formularios HTML? Explíquese qué ocurre durante el tiempo de vida de un servlet. (**Apartado 7.7.3**)
- ¿Cuál es la diferencia entre servlets y JSP? ¿Cuándo se deberían usar servlets y cuándo JSP? (**Apartado 7.7.4**)
- ¿Por qué es necesario mantener el estado en la capa intermedia? ¿Qué son las cookies? ¿Cómo maneja las cookies un navegador? ¿Cómo se puede acceder a los datos en cookies desde servlets? (**Apartado 7.7.5**)

EJERCICIOS

Ejercicio 7.1 Contéstense brevemente a las siguientes preguntas:

1. Explíquense los siguientes términos y descríbase para qué se utilizan: HTML, URL, XML, Java, JSP, XSL, XSLT, servlet, cookie, HTTP, CSS, DTD.
2. ¿Qué es CGI? ¿Por qué se introdujo CGI? ¿Cuáles son las desventajas de una arquitectura que utilice secuencias de comandos CGI?
3. ¿Cuál es la diferencia entre un servidor Web y un servidor de aplicaciones? ¿Qué funcionalidad proporcionan los servidores de aplicaciones normalmente?
4. ¿Cuándo un documento XML está bien formado? ¿Cuándo es válido?

Ejercicio 7.2 Contéstense brevemente las siguientes preguntas sobre el protocolo HTTP:

1. ¿Qué es un protocolo de comunicación?
2. ¿Cuál es la estructura de un mensaje de solicitud HTTP? ¿Cuál es la estructura de un mensaje de respuesta HTTP? ¿Por qué los mensajes HTTP incluyen un campo versión?
3. ¿Qué es un protocolo sin estado? ¿Por qué HTTP fue diseñado sin estado?
4. Muéstrese el mensaje de solicitud HTTP generado cuando se solicita la página de inicio de este libro (<http://www.cs.wisc.edu/~dbbook>). Muéstrese el mensaje de respuesta HTTP que el servidor genera para esta página.

Ejercicio 7.3 En este ejercicio se pide escribir la funcionalidad de una cesta de compra genérica; se utilizará en varios ejercicios posteriores. Escríbase un conjunto de páginas JSP que muestren una cesta de compra de artículos y permitan a los usuarios añadir, eliminar y modificar el número de artículos. Para ello, utilícese un esquema de almacenamiento de cookies que guarde la siguiente información:

- El identificador del usuario propietario de la cesta de compra.
- El número de productos almacenados en la cesta de compra.

- El identificador y la cantidad de cada producto.

Cuando se manipulen cookies hay que asignar la propiedad **Expires** de modo que la cookie persista durante una sesión o indefinidamente. Experimente con cookies utilizando JSP para asegurarse de saber cómo recuperar, asignar valores y borrar la cookie.

Es necesaria la creación de cinco páginas JSP para completar el prototipo:

- **Página de inicio (index.jsp)**: éste es el punto de entrada principal. Tiene un enlace que dirige al usuario a la página de productos para que pueda empezar a comprar.
- **Página de productos (productos.jsp)**: muestra una lista con todos los productos en la base de datos con sus descripciones y precios. Ésta es la página principal donde el usuario llena la cesta de compra. Cada producto listado debe tener un botón a su lado que permita añadirlo a la cesta de compra. (Si el artículo ya está en la cesta de compra, incrementa la cantidad en uno.) También debe haber un contador que muestre el número total de artículos que hay actualmente en la cesta de compra. Obsérvese que si un usuario tiene una cantidad de cinco unidades del mismo artículo en la cesta, el contador debe indicar una cantidad total de cinco. La página también contiene un botón que dirige al usuario a la página de la cesta.
- **Página de la cesta (cesta.jsp)**: muestra una lista con todos los artículos en la cookie de la cesta de compra. El listado de cada artículo debe incluir el nombre del producto, el precio, un cuadro de texto para la cantidad (el usuario puede cambiar aquí la cantidad de artículos) y un botón para eliminar el artículo de la cesta de compra. Esta página tiene otros tres botones: un botón para continuar comprando (que devuelve al usuario a la página de productos), un segundo botón para actualizar la cookie con las cantidades modificadas en los cuadros de texto, y un tercer botón para confirmar el pedido, que dirige al usuario a la página de confirmación.
- **Página de confirmación (confirmar.jsp)**: lista el pedido final. Esta página tiene dos botones. Un botón cancela el pedido y el otro envía el pedido completado. El botón de cancelación borra la cookie y devuelve al usuario a la página de inicio. El botón de envío actualiza la base de datos con el nuevo pedido, borra la cookie y devuelve al usuario a la página de inicio.

Ejercicio 7.4 En el ejercicio anterior se debe sustituir la página **productos.jsp** con la siguiente *página de búsqueda busqueda.jsp*. Esta página permite a los usuarios buscar productos por nombre o descripción. Debe contener un cuadro de texto para el texto de búsqueda y botones de radio para permitir al usuario elegir entre búsqueda por nombre o por descripción (así como un botón de envío para obtener los resultados). La página que maneja los resultados debe basarse en **productos.jsp** (como se describe en el ejercicio anterior) y llamarse **busqueda.jsp**. Debe recuperar todos los registros en los que el texto de búsqueda es una subcadena del nombre o la descripción (según haya elegido el usuario). Para integrar esto con el ejercicio anterior simplemente hay que sustituir todos los enlaces a **productos.jsp** por **busqueda.jsp**.

Ejercicio 7.5 Escríbase un mecanismo de autenticación simple (sin transferir claves cifradas, para simplificar). Se dice que un usuario está autenticado si ha proporcionado una combinación usuario-clave válida para el sistema; de lo contrario, se dice que no está autenticado. Supóngase para simplificar que se dispone de un esquema de base de datos que almacena solamente el identificador de cliente y la clave:

```
claves(idc: integer, nombreusuario: string, clave: string)
```

1. ¿Cómo y dónde se va a seguir la pista cuando un usuario haya iniciado sesión en el sistema?
2. Diséñese una página que permita iniciar sesión en el sistema a un usuario registrado.
3. Diséñese una cabecera de página que comprueba si el usuario que visita esta página ha iniciado sesión.

Ejercicio 7.6 (Por Jeff Derstadt) LibrosTécnicos.com está en proceso de reorganización de su sitio Web. Un problema principal es cómo manejar eficientemente un gran número de resultados de búsqueda. En un estudio de interacción humana, se encontró que los usuarios de módem normalmente les gusta ver 20 resultados de

búsqueda a la vez, y les gustaría programar esta lógica en el sistema. Las consultas que devuelven lotes de resultados ordenados se denominan *consultas de los N primeros*. (Véase el Apartado 16.5 para obtener más información sobre soporte de base de datos para consultas de los N primeros.) Por ejemplo, se devuelven los resultados 1-20, después los resultados 21-40, después 41-60, etcétera. Se utilizan distintas técnicas para realizar consultas de los N primeros y a LibrosTécnicos.com le gustaría que implementara dos de ellas.

Infraestructura. Créese una base de datos con una tabla llamada Libros e introduzcanse en ella algunos libros, utilizando el formato siguiente. Esto producirá 111 libros en su base de datos con títulos AAA, BBB, CCC, DDD o EEE, pero las claves no son secuenciales para los libros con el mismo título.

Libros(*idlibro*: INTEGER, *título*: CHAR(80), *autor*: CHAR(80), *precio*: REAL)

```
For i = 1 to 111 {
    Insertar la tupla (i, "AAA", "AAA Autor", 5.99)
    i = i + 1
    Insertar la tupla (i, "BBB", "BBB Autor", 5.99)
    i = i + 1
    Insertar la tupla (i, "CCC", "CCC Autor", 5.99)
    i = i + 1
    Insertar la tupla (i, "DDD", "DDD Autor", 5.99)
    i = i + 1
    Insertar la tupla (i, "EEE", "EEE Autor", 5.99)
}
```

Técnica del marcador de posición. El enfoque más simple a las consultas de los N primeros es almacenar un marcador de posición para la primera y última tuplas resultado y después realizar la misma consulta. Cuando se devuelven los resultados de la consulta, se puede iterar sobre los marcadores de posiciones y devolver los 20 resultados anteriores o posteriores.

Tuplas mostradas	Posición inferior	Conjunto anterior	Posición superior	Siguiente conjunto
1-20	1	Ninguno	20	21-40
21-40	21	1-20	40	41-60
41-60	41	21-40	60	61-80

Escríbase una página Web en JSP que muestre el contenido de la tabla Libros, ordenada por Título e IdLibro, y que muestre 20 resultados cada vez. Debe haber un enlace (donde sea apropiado) para obtener los 20 resultados anteriores o los 20 siguientes. Para ello se pueden codificar los marcadores de posición en los enlaces Anterior y Siguiente de la siguiente forma. Supóngase que están mostrando los registros 21-40. Entonces, el enlace a Anterior es `display.jsp?lower=21` y el enlace a Siguiente es `display.jsp?upper=40`.

No se debe mostrar un enlace a Anterior cuando no haya resultados anteriores; tampoco se debe mostrar un enlace a Siguiente si no hay más resultados. Cuando la página sea llamada de nuevo para obtener otro lote de resultados, se puede realizar la misma consulta para obtener todos los registros, iterando a lo largo del conjunto de resultados hasta alcanzar el punto de inicio apropiado, y a continuación mostrar los 20 nuevos resultados.

¿Cuáles son las ventajas e inconvenientes de esta técnica?

Técnica de restricción de consulta. Una segunda técnica para realizar las consultas de los N primeros es imponer una restricción de límite en la consulta (en la cláusula WHERE) de manera que la consulta devuelva sólo los resultados que no se han mostrado todavía. Aunque esto cambia la consulta, se devuelven menos resultados y ahorra el coste de iterar hasta el límite inferior. Por ejemplo, considérese la siguiente tabla, ordenada por (título, clave primaria).

Lote	Resultado número	Título	Clave primaria
1	1	AAA	105
1	2	BBB	13
1	3	CCC	48
1	4	DDD	52
1	5	DDD	101
2	6	DDD	121
2	7	EEE	19
2	8	EEE	68
2	9	FFF	2
2	10	FFF	33
3	11	FFF	58
3	12	FFF	59
3	13	GGG	93
3	14	HHH	132
3	15	HHH	135

En el lote 1 se muestran las filas 1 a 5, en el lote 2 las filas 6 a 10, y así sucesivamente. Utilizando la técnica del marcador de posición se devolverían los 15 resultados para cada lote. Utilizando la técnica de restricción, el lote 1 muestra los resultados 1-5 pero devuelve los resultados 1-15, el lote 2 muestra los resultados 6-10 pero devuelve solamente los resultados 6-15, y el lote 3 muestra los resultados 11-15 y devuelve estos mismos resultados.

La restricción puede agregarse a la consulta debido a que la tabla está ordenada. Considérese la siguiente consulta del lote 2 (que muestra los resultados 6-10):

```
EXEC SQL SELECT L.Título
  FROM Libros L
 WHERE (L.Título = 'DDD' AND L.Idlibro > 101) OR (L.Título > 'DDD')
 ORDER BY L.Título, L.Idlibro
```

Esta consulta selecciona primero todos los libros con el título ‘DDD’, pero con clave primaria mayor que el registro 5 (que tiene un valor de clave primaria 101). Esto devuelve el registro 6. Además, también se devuelve cualquier libro que tenga un título alfabéticamente mayor a ‘DDD’. Por tanto pueden mostrarse los primeros cinco resultados.

Se debe conservar la siguiente información para que los botones Anterior y Siguiente devuelvan más resultados:

- **Anterior:** el título del *primer* registro del conjunto anterior, y la clave primaria del *primer* registro del conjunto anterior.
- **Siguiente:** el título del *primer* registro del siguiente conjunto; la clave primaria del *primer* registro del conjunto siguiente.

Estos cuatro elementos de información pueden codificarse en los botones Anterior y Siguiente como se ha visto en la parte anterior. Utilizando la tabla de base de datos de la primera parte, escribase una página JavaServer que muestre la información de libros de 20 en 20 registros. La página debe incluir los botones *Anterior* y *Siguiente* para mostrar el conjunto anterior o siguiente si hay alguno. Empléese la consulta de restricción para obtener dichos conjuntos.

EJERCICIOS BASADOS EN PROYECTOS

En este capítulo se continuará con los ejercicios del capítulo anterior y se crearán las partes de la aplicación que residen en la capa intermedia y en la capa de presentación. Se puede encontrar más información sobre estos ejercicios y material para otros ejercicios en:



<http://www.cs.wisc.edu/~dbbook>

Ejercicio 7.7 Recuérdese el sitio Web de Discos Simpueblo (Notown Records) sobre el que trabajó en el Ejercicio 6.6. A continuación se pide desarrollar las páginas reales del sitio Web de Discos Simpueblo. Diseñese la parte del sitio Web que involucre la capa de presentación y la capa intermedia, e intégruese el código que se escribió en el Ejercicio 6.6 para acceder a la base de datos.

1. Describíbase en detalle el conjunto de páginas Web que los usuarios pueden acceder. Téngase en cuenta lo siguiente:
 - Todos los usuarios empiezan en una página común.
 - Para cada acción, ¿qué entrada proporciona el usuario? ¿Cómo la proporciona: pulsando un enlace o a través de un formulario HTML?
 - ¿Por qué secuencia de pasos pasa un usuario para comprar un disco? Describíbase el flujo de alto nivel de la aplicación mostrando cómo se maneja cada acción del usuario.
2. Escríbanse las páginas Web en HTML sin contenido dinámico.
3. Escríbanse una página que permita a los usuarios iniciar sesión en el sitio Web. Empléense cookies para almacenar la información permanentemente en el navegador del usuario.
4. Añádase a la página de inicio de sesión código JavaScript que compruebe que el nombre de usuario sólo está formado por caracteres de la a a la z.
5. Añádase a las páginas que permiten a los usuarios almacenar artículos en una cesta de compra una condición que compruebe si el usuario ha iniciado sesión en el sitio. Si todavía no la ha iniciado, debería ser imposible añadir artículos a la cesta de compra. Implementar esta funcionalidad mediante JSP comprobando la información de cookie del usuario.
6. Créense las páginas restantes para terminar el sitio Web.

Ejercicio 7.8 Recuérdese el proyecto de farmacia en línea del Ejercicio 6.7 del Capítulo 6. Síganse los mismos pasos del Ejercicio 7.7 para diseñar las capas de lógica de la aplicación y presentación, y termínese el sitio Web.

Ejercicio 7.9 Recuérdese el proyecto de farmacia en línea del Ejercicio 6.8 del Capítulo 6. Síganse los mismos pasos del Ejercicio 7.7 para diseñar las capas de lógica de la aplicación y presentación, y termínese el sitio Web.

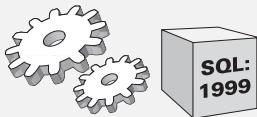
Ejercicio 7.10 Recuérdese el proyecto de farmacia en línea del Ejercicio 6.9 del Capítulo 6. Síganse los mismos pasos del Ejercicio 7.7 para diseñar las capas de lógica de la aplicación y presentación, y termínese el sitio Web.

NOTAS BIBLIOGRÁFICAS

La última versión de los estándares mencionados en este capítulo puede encontrarse en el sitio Web del World Wide Web Consortium (www.w3.org). Contiene enlaces a información sobre HTML, hojas de estilo en cascada, XML, XSL y mucho más. El libro de Hall es una introducción a las tecnologías de programación Web [232]; un buen punto de partida en Web es www.Webdeveloper.com. Hay muchos libros introductorios sobre programación CGI, por ejemplo [142, 130]. La página de JavaSoft (java.sun.com) es un buen punto de inicio para servlets, JSP y todas las demás tecnologías relacionadas con Java. El libro de Hunter [258] es una buena introducción a los servlets Java. Microsoft soporta páginas de servidor activo (Active Server Pages, ASP), una tecnología comparable a JSP. Se puede encontrar más información sobre ASP en la página de la red de desarrolladores de Microsoft (msdn.microsoft.com).

Se pueden encontrar excelentes sitios Web dedicados al fomento de XML, por ejemplo www.xml.com y www.ibm.com/xml, que también contienen un gran número de enlaces con información sobre otros estándares. Hay buenos libros introductorios sobre muchos aspectos de XML, por ejemplo [127, 97, 359, 297, 253, 211]. Se puede encontrar información sobre UNICODE en su página principal <http://www.unicode.org>.

Se puede encontrar información sobre páginas JavaServer y servlets en la página principal de JavaSoft en java.sun.com, java.sun.com/products/jsp y en java.sun.com/products/servlet.



8

VISIÓN GENERAL DE LA GESTIÓN DE TRANSACCIONES

- ¿Qué cuatro propiedades de transacciones garantiza un SGBD?
- ¿Por qué un SGBD entrelaza transacciones?
- ¿Cuál es el criterio de corrección de la ejecución entrelazada?
- ¿Qué tipos de anomalías pueden producir las transacciones entrelazadas?
- ¿Cómo utiliza los bloqueos un SGBD para asegurar un entrelazamiento correcto?
- ¿Cuál es el impacto del bloqueo en el rendimiento?
- ¿Qué comandos SQL permiten a los programadores seleccionar las características de las transacciones y reducir la sobrecarga por el bloqueo?
- ▶ **Conceptos clave:** propiedades ACID, atomicidad, consistencia, aislamiento, durabilidad; planificaciones, secuencialidad, recuperabilidad, evitar abortos en cascada; anomalías, lecturas desfasadas, lecturas no repetibles, actualizaciones perdidas; protocolos de bloqueo, bloqueos exclusivos y compartidos, bloqueo estricto en dos fases; rendimiento del bloqueo, pugna, zonas calientes; características de las transacciones SQL, puntos de almacenamiento, retrocesos, fantasmas, modo de acceso, nivel de aislamiento; gestor de transacciones.

Siempre digo: mantén un diario y algún día él te mantendrá a ti.

— Mae West

En este capítulo se trata el concepto de *transacción*, que es el fundamento de la ejecución concurrente y la recuperación de un fallo del sistema en un SGBD. Una transacción se define como *cualquier ejecución* de un programa de usuario en un SGBD y difiere de una ejecución de un programa fuera del SGBD (por ejemplo, un programa C ejecutándose sobre UNIX) en

aspectos importantes. (La ejecución del mismo programa varias veces genera varias transacciones.)

Por motivos de rendimiento, un SGBD tiene que entrelazar las acciones de varias transacciones. (El entrelazamiento de transacciones se motiva en detalle en el Apartado 8.3.1.) No obstante, para proporcionar a los usuarios una forma simple de entender el efecto de ejecutar sus programas, el entrelazamiento se realiza cuidadosamente para asegurar que el resultado de una ejecución concurrente de transacciones sea sin embargo equivalente (en su efecto sobre la base de datos) a alguna ejecución secuencial, una cada vez, del mismo conjunto de transacciones. La gestión de ejecuciones concurrentes de un SGBD es un aspecto importante de la gestión de transacciones, y es el objeto del *control de concurrencia*. Un problema estrechamente relacionado es la gestión de transacciones parciales de un SGBD, transacciones que son interrumpidas antes de terminar normalmente. El SGBD asegura que los cambios hechos por tales transacciones parciales no son observables por otras transacciones, lo cual es el objeto de la *recuperación de caídas del sistema*. En este capítulo se proporciona una amplia introducción al control de concurrencia en un SGBD.

En el Apartado 8.1 se tratan cuatro propiedades fundamentales de las transacciones de base de datos y la forma en que los SGBD las garantizan. En el Apartado 8.2 se presenta una forma abstracta de describir una ejecución entrelazada de varias transacciones, denominada *planificación*. El Apartado 8.3 trata diversos problemas que pueden producirse debido a la ejecución entrelazada. El control de concurrencia basado en bloqueos, el enfoque más ampliamente utilizado, se introduce en el Apartado 8.4. Los problemas de rendimiento asociados al control de concurrencia basado en bloqueos se tratan en el Apartado 8.5. Finalmente, en el Apartado 8.6 se consideran las propiedades de los bloqueos y las transacciones en el contexto de SQL.

8.1 LAS PROPIEDADES ACID

El concepto de transacción de base de datos se introdujo en el Apartado 1.7. Recapitulando brevemente, una transacción es una ejecución de un programa de usuario, visto por el SGBD como una serie de operaciones de lectura y escritura.

Un SGBD debe garantizar cuatro propiedades importantes de las transacciones para mantener los datos en presencia de accesos concurrentes y fallos del sistema:

1. Los usuarios deben poder considerar la ejecución de cada transacción como **atómica**: o se realizan todas las acciones o no se realiza ninguna. Los usuarios no deben preocuparse del efecto de transacciones incompletas (por ejemplo, cuando ocurre una caída del sistema).
2. Cada transacción, ejecutada por sí misma sin la ejecución concurrente de otras transacciones, debe preservar la **consistencia** de la base de datos. El SGBD supone que se mantiene la consistencia de cada transacción. Garantizar esta propiedad en una transacción es responsabilidad del usuario.
3. Los usuarios deben poder comprender una transacción sin considerar el efecto de otras transacciones ejecutándose concurrentemente, incluso si el SGBD entrelaza las acciones de varias transacciones por motivos de rendimiento. Esta propiedad se denomina a veces

aislamiento: las transacciones están aisladas, o protegidas, de los efectos de la planificación concurrente de otras transacciones.

- Una vez que el SGBD informa al usuario de que una transacción se ha completado satisfactoriamente, sus efectos deben persistir incluso si se produce una caída del sistema antes de que todos los cambios sean reflejados en disco. Esta propiedad se denomina **durabilidad**.

El acrónimo ACID se utiliza a veces para referirse a estas cuatro propiedades de las transacciones: atomicidad, consistencia, aislamiento (*isolation* en inglés, N. del T.) y durabilidad. Ahora se considerará la forma en que un SGBD garantiza estas propiedades.

8.1.1 Consistencia y aislamiento

Los usuarios son responsables de asegurar la consistencia de una transacción. Es decir, el usuario que envía una transacción debe asegurar que, cuando se ejecute con un ejemplar “consistente” de la base de datos, dejará a su vez la base de datos en un estado “consistente” cuando termine. Por ejemplo, el usuario puede (naturalmente) tener el criterio de consistencia de que las transferencias de fondos entre cuentas de un banco no deben cambiar la cantidad de dinero total en las cuentas. Para transferir dinero de una cuenta a otra, una transacción debe cargar el importe a una cuenta, dejando la base de datos temporalmente inconsistente en sentido global, aun cuando el nuevo balance de la cuenta satisface cualquier restricción de integridad respecto al rango de balances de cuenta aceptables. La noción de base de datos consistente del usuario se preserva cuando a la segunda cuenta se le abona la cantidad transferida. Si un programa de transferencia defectuoso siempre abona a la segunda cuenta un euro menos de la cantidad cargada en la primera cuenta, no puede esperarse que el SGBD detecte inconsistencias debido a este tipo de errores en la lógica del programa de usuario.

La propiedad de aislamiento se asegura garantizando que, aunque las acciones de varias transacciones puedan estar entrelazadas, el efecto neto es idéntico a ejecutar todas las transacciones una detrás de otra en algún orden secuencial. (Se verá la implementación de esta garantía en un SGBD en el Apartado 8.4.) Por ejemplo, si dos transacciones T_1 y T_2 se ejecutan concurrentemente, se garantiza el efecto neto equivalente a ejecutar (todo) T_1 seguido de la ejecución de T_2 , o ejecutar T_2 seguido de la ejecución de T_1 . (El SGBD no garantiza el orden efectivamente elegido.) Si cada transacción hace corresponder un ejemplar consistente de la base de datos a otra ejemplar consistente, ejecutar varias transacciones una después de otra (a partir de un ejemplar consistente inicial de la base de datos) resulta en un ejemplar final de la base de datos consistente.

La **consistencia de base de datos** es la propiedad por la que toda transacción ve un ejemplar consistente de la base de datos. La consistencia de la base de datos se obtiene de la atomicidad, aislamiento y consistencia de las transacciones. A continuación se tratará la forma en que un SGBD garantiza la atomicidad y durabilidad.

8.1.2 Atomicidad y durabilidad

Las transacciones pueden ser incompletas por tres tipos de razones. En primer lugar, una transacción puede ser **abortada**, o finalizada insatisfactoriamente, por el SGBD porque ocu-

rre alguna anomalía durante su ejecución. Si una transacción es abortada, es reiniciada automáticamente y ejecutada de nuevo. En segundo lugar, el sistema puede fallar (por ejemplo, porque la alimentación eléctrica se interrumpa) mientras una o varias transacciones están ejecutándose. En tercer lugar, una transacción puede encontrar una situación inesperada (por ejemplo, leer un valor inesperado de un dato o ser incapaz de acceder a algún disco) y decidir abortar (es decir, finalizar por sí misma).

Por supuesto, como los usuarios piensan que las transacciones son atómicas, una transacción que se interrumpe en la mitad puede dejar la base de datos en un estado inconsistente. Por tanto, el SGBD debe encontrar la forma de eliminar los efectos de transacciones parciales de la base de datos. Es decir, debe asegurar la atomicidad de la transacción: o se llevan a cabo todas las acciones de la transacción, o no se realiza ninguna. Un SGBD asegura la atomicidad de las transacciones *deshaciendo* las acciones de las transacciones incompletas. Esto significa que los usuarios pueden ignorar las transacciones incompletas cuando piensan las modificaciones que se realizan en la base de datos por las transacciones a lo largo del tiempo. Para poder hacer esto, el SGBD mantiene un registro, denominado *registro histórico*, de todas las escrituras en la base de datos. El registro histórico también se utiliza para asegurar la durabilidad: si el sistema falla antes de que se escriban en disco los cambios realizados por una transacción finalizada, se utiliza el registro histórico para recordar y restaurar estos cambios cuando el sistema reinicie.

8.2 TRANSACCIONES Y PLANIFICACIONES

Una transacción es vista por el SGBD como una serie, o *lista*, de **acciones**. Las acciones que pueden ejecutarse en una transacción incluyen **lecturas** y **escrituras** de *objetos de base de datos*. Para hacer la notación sencilla, se supone que un objeto O siempre se lee en una variable de programa que también se denomina O . Por tanto, se puede denotar la acción de una transacción T que lee un objeto O como $L_T(O)$; de forma similar, se puede denotar la escritura como $E_T(O)$. Cuando una transacción T está clara por el contexto, se omitirá el subíndice.

Además de leer y escribir, cada transacción *debe* especificar como última acción bien **comprometer** (es decir, la transacción está completada satisfactoriamente) o **abortar** (la transacción está terminada y deshechas todas las acciones realizadas hasta el momento). $Abortar_T$ representa la acción de abortar T , y $Comprometer_T$ la de comprometer T .

Hay dos suposiciones importantes:

1. Las transacciones interactúan mutuamente *sólo* a través de operaciones de lectura y escritura de la base de datos; por ejemplo, no se permite el intercambio de mensajes.
2. Una base de datos es una colección *fija* de objetos *independientes*. Cuando se añaden o eliminan objetos de una base de datos o existen relaciones entre objetos de la base de datos que se quieren aprovechar por rendimiento, surgen problemas adicionales.

Si se viola la primera suposición, el SGBD no tiene forma de detectar o prevenir inconsistencias causadas por estas interacciones externas entre transacciones, y es responsabilidad del programador de la aplicación asegurar que el programa se comporte correctamente. Se relajará la segunda suposición en el Apartado 8.6.2.

Una **planificación** es una lista de acciones (lectura, escritura, abortar o comprometer) de un conjunto de transacciones, y el orden en el que dos acciones de una transacción T aparecen en una planificación debe ser el mismo que el orden en el que aparecen en T . Intuitivamente, una planificación representa una secuencia de ejecución real o potencial. Por ejemplo, la planificación de la Figura 8.1 muestra un orden de ejecución de las acciones de dos transacciones $T1$ y $T2$. Se avanza en el tiempo de arriba hacia abajo de una fila a la otra. Es importante resaltar que una planificación describe las acciones de transacciones *según las*

$T1$	$T2$
$L(A)$	
$E(A)$	
	$L(B)$
	$E(B)$
$L(C)$	
$E(C)$	

Figura 8.1 Una planificación que implica dos transacciones

ve el SGBD. Además de estas acciones, una transacción puede realizar otras, tales como leer o escribir en archivos del sistema operativo, evaluar expresiones aritméticas, etcétera; sin embargo, se supone que estas acciones no afectan a otras transacciones, es decir, el efecto de una transacción sobre otra puede entenderse solamente en términos de los objetos comunes de la base de datos que leen y escriben.

Obsérvese que la planificación de la Figura 8.1 no contiene ninguna acción de abortar o comprometer para ninguna de las transacciones. Una planificación que contenga una de estas acciones para cada transacción se denomina una **planificación completa**. Una planificación completa debe contener todas las acciones de cada transacción que aparezca en ella. Si las acciones de las diferentes transacciones no están entrelazadas —es decir, las transacciones se ejecutan desde el principio hasta el final, una por una— la planificación se denomina **planificación secuencial**.

8.3 EJECUCIÓN CONCURRENTE DE TRANSACCIONES

Ahora que se ha introducido el concepto de planificación, se dispone de la forma conveniente de describir ejecuciones entrelazadas de transacciones. Los SGBD entrelazan las acciones de distintas transacciones para mejorar el rendimiento, pero no todos los entrelazamientos posibles pueden permitirse. En este apartado se consideran los entrelazamientos, o planificaciones, que un SGBD debería permitir.

8.3.1 Motivación para la ejecución concurrente

La planificación mostrada en la Figura 8.1 representa una ejecución entrelazada de dos transacciones. Asegurar el aislamiento de las transacciones mientras se permite esta ejecución

concurrente es difícil pero necesario por razones de rendimiento. En primer lugar, mientras que una transacción está esperando a que se lea una página de disco, la CPU puede procesar otra transacción. Esto es así porque la actividad de E/S puede hacerse en paralelo con la actividad de la CPU de un ordenador. Solapar la actividad de E/S y de la CPU reduce la cantidad de tiempo en que los discos y los procesadores están libres e incrementa la **productividad del sistema** (el número medio de transacciones completadas en un tiempo dado). En segundo lugar, la ejecución entrelazada de una transacción corta con una transacción larga permite normalmente que la transacción corta termine rápidamente. En la ejecución secuencial, una transacción corta podría quedar bloqueada detrás de una transacción larga, llevando a retrasos impredecibles en el **tiempo de respuesta**, el tiempo medio que lleva completar una transacción.

8.3.2 Secuencialidad

Una **planificación secuenciable** sobre un conjunto S de transacciones comprometidas es una planificación cuyo efecto sobre cualquier ejemplar consistente de la base de datos se garantiza idéntico al de alguna planificación secuencial sobre S . Es decir, el ejemplar de la base de datos que resulta de ejecutar la planificación es idéntico al ejemplar que resulta de ejecutar las transacciones en *algún* orden secuencial¹.

Como ejemplo, la planificación mostrada en la Figura 8.2 es secuenciable. Aun cuando las acciones de $T1$ y $T2$ se entrelazan, el resultado de esta planificación es equivalente a ejecutar $T1$ (completamente) y después $T2$. Intuitivamente, la lectura y escritura de B en $T1$ no está afectada por las acciones de $T2$ sobre A , y el efecto neto es el mismo que si estas acciones se “intercambiaron” para obtener la planificación $T1; T2$.

$T1$	$T2$
$L(A)$	
$E(A)$	
	$L(A)$
	$E(A)$
$L(B)$	
$E(B)$	
	$L(B)$
	$E(B)$
	Comprometer
Comprometer	

Figura 8.2 Una planificación secuenciable

La ejecución secuencial de transacciones en diferentes órdenes puede producir distintos resultados, pero todos se suponen aceptables; el SGBD no garantiza cuál de ellos será el resultado de una ejecución entrelazada. Para ver esto, obsérvese que las dos transacciones

¹Si una transacción imprime un valor en la pantalla, este “efecto” no es capturado directamente en la base de datos. Por simplicidad, se supone que estos valores también se escriben en la base de datos.

de ejemplo de la Figura 8.2 pueden entrelazarse como se muestra en la Figura 8.3. Esta planificación, también secuenciable, es equivalente a la planificación secuencial $T2; T1$. Si $T1$ y $T2$ son enviadas concurrentemente a un SGBD, se podría elegir entre cualquiera de las dos planificaciones (entre otras).

$T1$	$T2$
	$L(A)$
$L(A)$	$E(A)$
	$L(B)$
$E(A)$	$E(B)$
$L(B)$	
$E(B)$	
Comprometer	Comprometer

Figura 8.3 Otra planificación secuenciable

La definición anterior de planificación secuenciable no cubre el caso de planificaciones que contienen transacciones abortadas. Se extiende esta definición para incluir este caso en el Apartado 8.3.4.

Finalmente, un SGBD podría algunas veces ejecutar transacciones de una forma no equivalente a ninguna ejecución secuencial; es decir, utilizando una planificación no secuenciable. Esto puede ocurrir por dos motivos. Primero, el SGBD podría utilizar un método de control de concurrencia que asegure que la planificación ejecutada, aunque no secuenciable en sí misma, sea equivalente a alguna planificación secuenciable. Por otro lado, SQL proporciona a los programadores de aplicaciones la posibilidad de ordenar al SGBD que elija planificaciones no secuenciables (véase el Apartado 8.6).

8.3.3 Anomalías debidas a la ejecución entrelazada

A continuación se ilustran las tres formas principales en las que una planificación que incluye dos transacciones que, manteniendo la consistencia y comprometidas, podrían dejar una base de datos consistente en un estado inconsistente. Dos acciones sobre el mismo objeto de datos **entran en conflicto** si por lo menos una de ellas es una escritura. Las tres situaciones anómalas pueden describirse respecto a cuándo las acciones de dos transacciones $T1$ y $T2$ entran en conflicto entre ellas: en un **conflicto de escritura-lectura (EL)**, $T2$ lee un objeto de datos escrito previamente por $T1$; los conflictos de **lectura-escritura (LE)** y de **escritura-escritura (EE)** se definen de forma similar.

Lectura de datos no comprometidos (conflictos EL)

La primera fuente de anomalías es que una transacción $T2$ pueda leer un objeto de base de datos A que ha sido modificado por otra transacción $T1$ que todavía no ha sido comprometida. Este tipo de lectura se denomina **lectura desfasada**. Un ejemplo sencillo ilustra la forma en que una planificación puede llevar a un estado inconsistente de la base de datos. Considérense dos transacciones $T1$ y $T2$, cada una de las cuales, ejecutadas independientemente, preservan la consistencia de la base de datos: $T1$ transfiere 100 € de A a B , y $T2$ incrementa tanto A como B en un 6% (por ejemplo, el interés anual depositado en estas dos cuentas). Supóngase que las acciones están entrelazadas de forma que (1) el programa de transferencia de cuentas $T1$ resta 100 € de la cuenta A , después (2) el programa de intereses de depósitos $T2$ lee el valor actual de las cuentas A y B y suma el 6% de interés a cada una, y después (3) el programa de transferencia de cuentas abona 100 € a la cuenta B . La planificación correspondiente, que es la vista que tiene el SGBD de esta serie de eventos, se ilustra en la Figura 8.4. El resultado de esta planificación es diferente de cualquier resultado posible ejecutando primero una de las transacciones y después la otra. El problema puede ligarse al hecho de que el valor de A escrito por $T1$ es leído por $T2$ antes de que $T1$ haya terminado todos sus cambios.

$T1$	$T2$
$L(A)$	
$E(A)$	
	$L(A)$
	$E(A)$
	$L(B)$
	$E(B)$
	Comprometer
$L(B)$	
$E(B)$	
Comprometer	

Figura 8.4 Lectura de datos no comprometidos

El problema general ilustrado aquí es que $T1$ puede escribir algún valor en A que hace la base de datos inconsistente. Siempre que $T1$ sobreesciba este valor con un valor “correcto” antes de comprometer, no se hace ningún daño si $T1$ y $T2$ se ejecutan en algún orden secuencial, porque $T2$ no verá la inconsistencia (temporal). Por otra parte, la ejecución entrelazada puede exponer esta inconsistencia y dar lugar a un estado final inconsistente de la base de datos.

Obsérvese que aunque una transacción debe dejar la base de datos en un estado consistente *después* de finalizar, no se requiere que mantenga la base de datos consistente mientras se está ejecutando. Este requerimiento sería demasiado restrictivo: para transferir dinero de una cuenta a otra, la transacción *debe* hacer el cargo en una cuenta, dejando la base de datos temporalmente inconsistente, y después el abono en la segunda cuenta, restaurando la consistencia.

Lecturas no repetibles (conflictos LE)

La segunda forma en la que puede producirse un comportamiento anómalo es cuando una transacción T_2 puede cambiar el valor de un objeto A que ha leído una transacción T_1 mientras se está ejecutando.

Si T_1 intenta leer el valor de A de nuevo, obtendrá un resultado diferente, aun cuando no ha modificado A en ese tiempo. Esta situación no puede producirse en una ejecución secuencial de dos transacciones; se denomina **lectura no repetible**.

Para ver por qué esto puede producir problemas, considérese el siguiente ejemplo. Supóngase que A es el número de copias disponibles de un libro. Una transacción que realice un pedido primero lee A , comprueba que es mayor a 0, y después lo decremente. La transacción T_1 lee A y ve el valor 1. La transacción T_2 también lee A y observa el valor 1, decremente A a 0 y compromete. La transacción T_1 entonces intenta decrementar A y obtiene un error (si hay una restricción de integridad que evita que A sea negativo).

Esta situación nunca puede producirse en una ejecución secuencial de T_1 y T_2 ; la segunda transacción leería A y vería 0 y por consiguiente no procedería con el pedido (y no intentaría decrementar A).

Sobreescritura de datos no comprometidos (conflictos EE)

La tercera fuente de comportamiento anómalo es que una transacción T_2 pueda sobreescribir el valor de un objeto A que ya ha sido modificado por una transacción T_1 , mientras T_1 todavía está ejecutándose. Aun si T_2 no lee el valor de A escrito por T_1 , existe un problema potencial, como se ilustra con el siguiente ejemplo.

Supóngase que Hernando y Fernando son dos empleados, y sus sueldos deben mantenerse iguales. La transacción T_1 asigna sus sueldos a 2.000 € y la transacción T_2 a 1.000 €. Si se ejecutan estas transacciones en el orden secuencial T_1 seguido de T_2 , ambos reciben el sueldo 1.000; el orden secuencial inverso da a cada uno el sueldo 2.000. Cualquiera de estos dos es aceptable desde un punto de vista de la consistencia (¡aunque Hernando y Fernando preferirían el sueldo mayor!). Obsérvese que ninguna de las dos transacciones lee el valor del sueldo antes de escribirlo — esta escritura se denomina **escritura ciega**, por razones obvias.

Ahora, considérese el siguiente entrelazamiento de las acciones de T_1 y T_2 : T_2 asigna el sueldo de Hernando a 1.000 €, T_1 asigna el sueldo de Fernando a 2.000 €, T_2 asigna el sueldo de Fernando a 1.000 € y compromete, y finalmente T_1 asigna el sueldo de Hernando a 2.000 € y compromete. El resultado no es idéntico al de ninguna de las dos posibles ejecuciones secuenciales, y la planificación entrelazada es por tanto no secuenciable. Incumple el criterio de consistencia por el que ambos sueldos deben ser iguales.

El problema es que sucede una **actualización perdida**. La primera transacción en comprometer, T_2 , sobreescribió el sueldo de Hernando como fue asignado por T_1 . En el orden secuencial T_2 seguido de T_1 , el sueldo de Fernando reflejaría la actualización de T_1 en lugar de la de T_2 , pero la actualización de T_1 “se pierde”.

8.3.4 Planificaciones que implican transacciones abortadas

En este apartado se extenderá la definición de secuencialidad para incluir transacciones abortadas². Intuitivamente, todas las acciones de las transacciones abortadas deben deshacerse, y es posible por tanto imaginar que nunca se realizaron. Utilizando esta intuición, se puede extender la definición de planificación secuenciable de la siguiente forma: una **planificación secuenciable** sobre un conjunto S de transacciones es una planificación en la que se garantiza que su efecto sobre cualquier ejemplar consistente de la base de datos será idéntico al de alguna planificación secuencial completa sobre el conjunto de transacciones *comprometidas* en S .

Esta definición de secuencialidad se basa en que las acciones de las transacciones abortadas se deshagan completamente, lo que puede ser imposible en algunas situaciones. Por ejemplo, supóngase que (1) una programa de transferencia de cuentas T_1 descuenta 100 € de la cuenta A , y después (2) un programa de intereses de depósitos T_2 lee los valores actuales de las cuentas A y B y añade un 6% de interés a cada una, compromete, y entonces (3) T_1 aborta. La planificación correspondiente se muestra en la Figura 8.5.

T_1	T_2
$L(A)$	
$E(A)$	
	$L(A)$
	$E(A)$
	$L(B)$
	$E(B)$
	Comprometer
Abortar	

Figura 8.5 Una planificación no recuperable

T_2 ha leído un valor de A que no debería haber estado nunca ahí. (Recuérdese que se supone que los efectos de las transacciones abortadas no son visibles a otras transacciones.) Si T_2 no se ha comprometido todavía, podríamos tratar la situación propagando el aborto de T_1 en cascada y abortar también T_2 ; este proceso aborta recursivamente cualquier transacción que lea datos escritos por T_2 , y así sucesivamente. Pero T_2 ya ha comprometido, y por tanto no es posible deshacer sus acciones. Se dice que este tipo de planificación es *no recuperable*. En una **planificación recuperable**, las transacciones se comprometen sólo después de que se comprometan a su vez todas las transacciones que hayan leído sus cambios. Si las transacciones leen solamente los cambios de transacciones comprometidas, la planificación no sólo es recuperable, sino que se puede abortar una transacción sin abortar otras transacciones en cascada. Este tipo de planificación se dice que **evita abortos en cascada**.

Existe otro problema potencial al deshacer las acciones de una transacción. Supóngase que una transacción T_2 sobreescribe el valor de un objeto A que ha sido modificado por

²También deben considerarse transacciones incompletas para un estudio riguroso de fallos del sistema, porque las transacciones que están activas cuando el sistema falla no están ni abortadas ni comprometidas. Sin embargo, y para este estudio informal, basta con considerar las planificaciones que implican transacciones comprometidas y abortadas.

una transacción T_1 , mientras que T_1 todavía está ejecutándose, y T_1 aborta posteriormente. Todos los cambios de T_1 a los objetos de la base de datos se deshacen restaurando el valor de cualquier objeto modificado al valor que tenía antes de los cambios de T_1 . Cuando T_1 aborta y sus cambios se deshacen de esta forma, también se pierden los cambios de T_2 , incluso si T_2 decide comprometerse. Así, por ejemplo, si A tenía originalmente el valor 5, cambiado a 6 por T_1 , y por T_2 a 7, si T_1 aborta, el valor de A vuelve a ser 5 de nuevo. Si T_2 se compromete, su cambio realizado en A se pierde inadvertidamente. Una técnica de control de concurrencia denominada B2F estricto, introducida en el Apartado 8.4, puede evitar este problema.

8.4 CONTROL DE CONCURRENCIA BASADO EN BLOQUEOS

Un SGBD debe ser capaz de asegurar que sólo se permiten planificaciones secuenciales y recuperables, y que no se pierden acciones de transacciones comprometidas cuando se deshacen los cambios de transacciones abortadas. Un SGBD normalmente utiliza un *protocolo de bloqueo* para conseguir esto. Un **bloqueo** es un pequeño objeto de contabilidad asociado con un objeto de la base de datos. Un **protocolo de bloqueo** es un conjunto de reglas a seguir por cada transacción (e impuestas por el SGBD) que asegura que, aun cuando las acciones de varias transacciones estén entrelazadas, el efecto neto sea idéntico a ejecutar todas en algún orden secuencial. Distintos protocolos de bloqueo utilizan diferentes tipos de bloqueos, tales como bloqueos compartidos o exclusivos, como se verá a continuación cuando se trate el protocolo B2F estricto.

8.4.1 Bloqueo en dos fases estricto (B2F estricto)

El protocolo de bloqueo más extensamente usado, denominado *bloqueo en dos fases estricto*, o *B2F estricto* (2-phase locking), tiene dos reglas. La primera es:

1. Si una transacción T quiere *leer* (respectivamente, *modificar*) un objeto, primero solicita un bloqueo **compartido** (respectivamente, **exclusivo**) sobre el objeto.

Por supuesto, una transacción que tiene un bloqueo exclusivo también puede leer el objeto; no se necesita un bloqueo compartido adicional. Una transacción que solicita un bloqueo se suspende hasta que el SGBD puede conceder el bloqueo. El SGBD lleva la cuenta de los bloqueos concedidos y asegura que si una transacción tiene un bloqueo exclusivo sobre un objeto, ninguna otra transacción tenga ningún bloqueo compartido ni exclusivo sobre el mismo objeto. La segunda regla de B2F estricto es:

2. Todos los bloqueos concedidos a una transacción se liberan cuando la transacción se completa.

El SGBD puede insertar automáticamente las peticiones para obtener y liberar bloqueos en las transacciones; los usuarios no necesitan preocuparse de estos detalles. (En el Apartado 8.6.3 se verá la forma en que los programadores de aplicaciones pueden seleccionar propiedades de transacciones y controlar la sobrecarga debida a los bloqueos.)

En efecto, el protocolo de bloqueo sólo permite entrelazamientos “seguros” de transacciones. Si dos transacciones acceden a partes completamente independientes de la base de

datos, obtienen concurrentemente los bloqueos que necesitan y siguen alegremente sus caminos. Por otra parte, si dos transacciones acceden al mismo objeto, y una quiere modificarlo, sus acciones son efectivamente ordenadas secuencialmente —todas las acciones de una de las transacciones (la que primero obtiene el bloqueo sobre el objeto común) se completan antes (de que este bloqueo se libere y) de que la otra transacción pueda continuar—.

Con $C_T(O)$ se denota la acción de una transacción T solicitando un bloqueo compartido (respectivamente, exclusivo) sobre el objeto O (respectivamente, $X_T(O)$) y se omite el subíndice de la transacción cuando queda claro por el contexto. Como ejemplo, considérese la planificación mostrada en la Figura 8.4. Este entrelazamiento podría resultar en un estado que no se puede producir con ninguna ejecución secuencial de las tres transacciones. Por ejemplo, $T1$ podría cambiar A de 10 a 20, después $T2$ (que lee el valor 20 de A) podría cambiar B de 100 a 200, y finalmente $T1$ leería el valor 200 de B . Si se ejecuta secuencialmente, bien $T1$ o $T2$ se ejecutaría primero, y leería los valores 10 de A y 100 de B : claramente, la ejecución entrelazada no es equivalente a ninguna de las ejecuciones secuenciales.

Si se utiliza el protocolo B2F estricto, este entrelazamiento no estaría permitido. Suponiendo que las transacciones se realizan a la misma velocidad relativa que antes, $T1$ obtendría primero un bloqueo exclusivo sobre A y entonces leería y escribiría A (Figura 8.6). Después,

$T1$	$T2$
$X(A)$	
$L(A)$	
$E(A)$	

Figura 8.6 Planificación que ilustra el protocolo B2F estricto

$T2$ solicitaría un bloqueo sobre A . Sin embargo, esta petición no puede concederse hasta que $T1$ libere su bloqueo exclusivo sobre A , y por consiguiente el SGBD suspende $T2$. $T1$ prosigue para obtener un bloqueo exclusivo sobre B , lee y escribe B , y finalmente compromete, liberando sus bloqueos. A continuación se concede la petición de bloqueo de $T2$, y éste prosigue su ejecución. En este ejemplo el protocolo de bloqueo resulta en una ejecución secuencial de las dos transacciones, mostrada en la Figura 8.7.

En general, no obstante, se pueden entrelazar las acciones de transacciones diferentes. Como ejemplo, considérese el entrelazamiento de dos transacciones mostrado en la Figura 8.8, que permite el protocolo B2F estricto.

Se puede demostrar que el algoritmo B2F estricto sólo permite planificaciones secuenciales. Ninguna de las anomalías tratadas en el Apartado 8.3.3 pueden producirse si el SGBD implementa el protocolo B2F estricto.

8.4.2 Interbloqueos

Considérese el siguiente ejemplo. La transacción $T1$ obtiene un bloqueo exclusivo sobre el objeto A , $T2$ obtiene un bloqueo exclusivo sobre B , $T1$ solicita un bloqueo exclusivo sobre B y se suspende, y $T2$ solicita un bloqueo exclusivo sobre A y se suspende. Ahora, $T1$ está esperando a que $T2$ libere su bloqueo y $T2$ está esperando a que $T1$ libere el suyo. Este tipo de ciclos de transacciones esperando la liberación de bloqueos se denominan **interbloqueos**.

$T1$	$T2$
$X(A)$	
$L(A)$	
$E(A)$	
$X(B)$	
$L(B)$	
$E(B)$	
Comprometer	
	$X(A)$
	$L(A)$
	$E(A)$
	$X(B)$
	$L(B)$
	$E(B)$
	Comprometer

Figura 8.7 Planificación que ilustra el protocolo B2F estricto con ejecución secuencial

$T1$	$T2$
$C(A)$	
$L(A)$	
	$C(A)$
	$L(A)$
	$X(B)$
	$L(B)$
	$E(B)$
	Comprometer
$X(C)$	
$L(C)$	
$E(C)$	
Comprometer	

Figura 8.8 Planificación que sigue B2F estricto con acciones entrelazadas

Claramente, estas dos transacciones no progresarán. Peor aún, mantendrán los bloqueos que pueden ser solicitados por otras transacciones. El SGBD debe prevenir o detectar (y resolver) estas situaciones de interbloqueo; el enfoque más común es detectarlos y resolverlos.

Una forma sencilla de identificar interbloqueos es utilizar un mecanismo de tiempo de espera. Si una transacción ha estado esperando demasiado tiempo por un bloqueo, se puede suponer (de forma pesimista) que está en un ciclo de interbloqueo y abortarla.

8.5 RENDIMIENTO DEL BLOQUEO

Los esquemas basados en bloqueos están diseñados para resolver conflictos entre transacciones, y utilizan dos mecanismos fundamentales: *blockear* y *abortar* transacciones. Ambos mecanismos implican una penalización en el rendimiento: las transacciones bloqueadas pueden contener a su vez bloqueos que fuerzan la espera de otras transacciones, y el hecho de abortar y reiniciar una transacción desperdicia obviamente el trabajo realizado por la transacción hasta ese momento. Un interbloqueo representa una instancia extrema de bloqueo en la que un conjunto de transacciones está bloqueado para siempre a no ser que el SGBD aborde una de ellas.

En la práctica, menos de un 1% de las transacciones están implicadas en un interbloqueo, y pocas transacciones se abortan por este motivo. Por consiguiente, la sobrecarga procede fundamentalmente de retrasos debidos al bloqueo de transacciones³. Considérese la forma en que los retardos por bloqueos afectan a la productividad del sistema. Las primeras transacciones probablemente no entrarán en conflicto, y la productividad del sistema crece proporcionalmente al número de transacciones activas. Conforme aumenta el número de transacciones que se ejecutan concurrentemente en el mismo número de objetos de base de datos, la probabilidad de que se bloqueen entre ellas crece. De este modo, los retrasos debidos a bloqueos se incrementan con el número de transacciones activas. De hecho, se llega a un punto en el que añadir otra transacción activa realmente reduce la productividad del sistema; esta transacción es bloqueada y compite con (y bloquea) las transacciones existentes. Se dice entonces que el sistema entra en **pugna** de transacciones, lo que se muestra en la Figura 8.9.

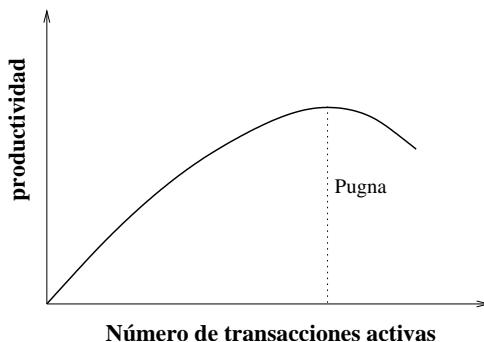


Figura 8.9 Pugna de transacciones por bloqueos

³Muchos interbloqueos comunes pueden evitarse mediante una técnica denominada *degradación de bloqueo*, implementada en la mayor parte de los sistemas comerciales.

Si un sistema de base de datos entra en pugna de transacciones, el administrador de la base de datos debería reducir el número de transacciones concurrentes permitidas. Se ha observado empíricamente que la pugna se produce cuando un 30% de las transacciones activas están bloqueadas, y un DBA debería monitorizar la fracción de transacciones bloqueadas para ver si el sistema está en riesgo de pugna.

La productividad del sistema se puede incrementar de tres formas (además de comprando un sistema más rápido):

- Bloqueando objetos lo más pequeños posible (reduciendo la probabilidad de que dos transacciones necesiten el mismo bloqueo).
- Reduciendo el tiempo que las transacciones mantienen los bloqueos (de forma que otras transacciones estén bloqueadas menos tiempo).
- Reduciendo las **zonas calientes**. Una zona caliente es un objeto de base de datos al que se accede y modifica frecuentemente, y que causa gran cantidad de retrasos por bloqueos. Las zonas calientes pueden afectar significativamente el rendimiento.

La granularidad del bloqueo está determinada principalmente por la implementación del bloqueo en el sistema de base de datos, y los programadores de aplicaciones y el DBA tienen poco control sobre ella. En el Apartado 13.10 se verá la mejora del rendimiento minimizando la duración de los bloqueos y utilizando técnicas para tratar las zonas calientes.

8.6 SOPORTE DE TRANSACCIONES EN SQL

Hasta ahora se han visto las transacciones y su gestión utilizando un modelo abstracto de una transacción como una secuencia de acciones de lectura, escritura y compromiso/aborts. Ahora se considerará el soporte que proporciona SQL a los usuarios para especificar el comportamiento en el nivel de transacciones.

8.6.1 Creación y terminación de transacciones

Una transacción se inicia automáticamente cuando un usuario ejecuta una sentencia que accede a la base de datos o a los catálogos, tal como una consulta SELECT, un comando UPDATE o una sentencia CREATE TABLE⁴.

Una vez que se ha iniciado una transacción, se pueden ejecutar otras sentencias como parte de esta transacción hasta que ésta finaliza con un comando COMMIT o ROLLBACK (la palabra clave de SQL para abortar la transacción).

En SQL:1999 se proporcionan dos nuevas características para aplicaciones que contengan transacciones de larga duración, o que deban ejecutar varias transacciones consecutivas. Para comprender estas extensiones, recuérdese que todas las acciones de una transacción determinada se ejecutan en orden, independientemente de que se entrelazan acciones de transacciones diferentes. Se puede considerar cada transacción como una secuencia de pasos.

⁴Algunas sentencias SQL —por ejemplo, la sentencia CONNECT, que conecta el programa de aplicación a un servidor de bases de datos— no requieren la creación de una transacción.

Transacciones anidadas en SQL:1999. El concepto de una transacción como una secuencia atómica de acciones se ha extendido en SQL:1999 mediante la introducción de los *puntos de almacenamiento*. Esto permite que se puedan deshacer selectivamente partes de una transacción. La introducción de los puntos de almacenamiento representa el primer soporte de SQL para el concepto de **transacciones anidadas**, que ha sido estudiado extensivamente por la comunidad de investigación. La idea es que una transacción puede tener varias subtransacciones anidadas, cada una de las cuales puede ser deshecha selectivamente. Los puntos de almacenamiento son una forma simple de anidamiento de un nivel.

La primera característica, denominada **punto de almacenamiento**, permite identificar un punto en una transacción y deshacer selectivamente las operaciones realizadas después de este punto. Esto es especialmente útil si la transacción realiza operaciones de tipo “qué pasa si”, y desea deshacer o mantener los cambios basándose en los resultados. Esto se puede conseguir definiendo puntos de almacenamiento.

En una transacción de larga duración, puede quererse definir una serie de puntos de almacenamiento. El comando **SAVEPOINT** permite dar a cada punto de almacenamiento un nombre:

SAVEPOINT *(nombre del punto de almacenamiento)*

Un comando **ROLLBACK** posterior puede especificar el punto de almacenamiento al que retroceder

ROLLBACK TO **SAVEPOINT** *(nombre del punto de almacenamiento)*

Si se definen tres puntos de almacenamiento *A*, *B* y *C* en ese orden, y después se retrocede a *A*, se deshacen todas las operaciones desde *A*, incluyendo la creación de los puntos de almacenamiento *B* y *C*. De hecho, el punto de almacenamiento *A* se deshace cuando se retrocede a él mismo, y debe re establecerse (a través de otro punto de almacenamiento) si se quiere retroceder a él de nuevo. Desde el punto de vista del bloqueo, los bloqueos obtenidos después de *A* pueden liberarse cuando se retrocede a *A*.

Resulta instructivo comparar el uso de puntos de almacenamiento con la alternativa de ejecutar una serie de transacciones (es decir, tratar todas las operaciones entre dos puntos de almacenamiento consecutivos como una nueva transacción). El mecanismo de puntos de almacenamiento ofrece dos ventajas. En primer lugar, se puede volver atrás sobre varios puntos de almacenamiento. En el enfoque alternativo, sólo se puede volver atrás a la transacción más reciente, que es equivalente a retroceder al punto de almacenamiento más reciente. En segundo lugar, se evita la sobrecarga de inicio de varias transacciones.

Aun usando puntos de almacenamiento, ciertas aplicaciones podrían requerir la ejecución de varias transacciones sucesivas. Para minimizar la sobrecarga en estos casos, SQL:1999 introduce otra característica, denominada **transacciones en cadena**. Se puede comprometer o deshacer una transacción e iniciar inmediatamente la siguiente. Esto se hace utilizando las palabras clave opcionales **AND CHAIN** en las sentencias **COMMIT** y **ROLLBACK**.

8.6.2 ¿Qué debería bloquearse?

Hasta ahora se han tratados las transacciones y el control de concurrencia en términos de un modelo abstracto en el que una base de datos contiene una colección fija de objetos, y cada transacción es una serie de operaciones de lectura y escritura sobre objetos individuales. Una pregunta importante a considerar en el contexto de SQL es lo que el SGBD debería tratar como un *objeto* cuando se crean los bloqueos de una sentencia SQL determinada (lo que es parte de una transacción).

Considérese la siguiente consulta:

```
SELECT M.categoría, MIN(M.edad)
FROM   Marineros M
WHERE  M.categoría = 8
```

Supongáse que esta consulta se ejecuta como parte de la transacción T_1 , y una sentencia SQL que modifica la edad de un marinero determinado, por ejemplo Juan, con $categoría=8$ se ejecuta como parte de la transacción T_2 . ¿Qué “objetos” debe bloquear el SGBD cuando se ejecutan estas transacciones? Intuitivamente, se debería detectar un conflicto entre estas transacciones.

El SGBD puede poner un bloqueo compartido en toda la tabla *Marineros* para T_1 y un bloqueo exclusivo en la misma tabla para T_2 , lo que aseguraría que las dos transacciones se ejecutan de forma secuenciable. Sin embargo, este enfoque produce una baja concurrencia, y se puede hacer mejor bloqueando objetos más pequeños, reflejando realmente lo que accede cada transacción. De esta forma, el SGBD puede hacer un bloqueo compartido en cada fila con $categoría=8$ para la transacción T_1 y un bloqueo exclusivo sólo en la fila modificada por la transacción T_2 . Así, otras transacciones de sólo lectura que no utilicen las filas con $categoría=8$ pueden acceder sin esperar a T_1 o T_2 .

Como ilustra este ejemplo, el SGBD puede bloquear objetos con diferente **granularidad**: se pueden bloquear tablas enteras o crear bloqueos de filas individuales. Este último enfoque es el tomado por los sistemas actuales porque ofrece mucho mejor rendimiento. En la práctica, mientras que el bloqueo por fila es generalmente mejor, la elección de la granularidad es complicada. Por ejemplo, una transacción que examine varias filas y modifique las que satisfacen alguna condición podrían ser mejor atendidas creando bloqueos compartidos en toda la tabla y bloqueos exclusivos en las filas a modificar.

Un segundo punto a tener en cuenta es que las sentencias SQL acceden conceptualmente a una colección de filas descritas por un *predicado de selección*. En el ejemplo anterior, la transacción T_1 accede a todas las filas con $categoría=8$. Se sugirió que esto podría tratarse creando bloqueos compartidos en todas las filas de *Marineros* con $categoría=8$. Desafortunadamente, esto es un poco simplista. Para ver por qué, considérese una sentencia SQL que inserte un nuevo marinero con $categoría=8$ y se ejecute como la transacción T_3 . (Obsérvese que este ejemplo viola la suposición de que hay un número fijo de objetos en la base de datos, pero evidentemente deben tratarse estas situaciones en la práctica.)

Supóngase que el SGBD crea bloqueos compartidos en cada fila existente de *Marineros* con $categoría=8$ para T_1 . Esto no evita que la transacción T_3 cree una nueva fila con $categoría=8$ y cree un bloqueo exclusivo en esta fila. Si esta nueva fila tiene un valor de *edad* menor a las demás, T_1 devuelve un valor que depende de cuándo fue ejecutada respecto a T_3 .

Sin embargo, el sistema de bloqueo propuesto no impone un orden relativo entre estas dos transacciones.

Este fenómeno se denomina el problema **fantasma**: una transacción recupera una colección de objetos (en términos de SQL, una colección de tuplas) dos veces y observa resultados diferentes, aun cuando la transacción no modifica ninguna de las tuplas. Para evitar los fantasmas, el SGBD debe bloquear conceptualmente *todas las posibles* filas con *categoria=8* en nombre de *T1*. Una forma de hacer esto es bloqueando toda la tabla, con el coste de la baja concurrencia. Es posible aprovechar los índices para hacerlo mejor, pero evitar los fantasmas puede tener en general un impacto significativo en la concurrencia.

Puede ocurrir que la aplicación que invoque *T1* pueda aceptar la inexactitud potencial debida a los fantasmas. Si es así, el enfoque de crear bloqueos compartidos para *T1* sobre las tuplas existentes es adecuado y ofrece mejor rendimiento. SQL permite a un programador hacer esta elección —y otras similares— explícitamente, como se verá a continuación.

8.6.3 Características de las transacciones en SQL

Con el objetivo de dar a los programadores el control sobre la sobrecarga producida por los bloqueos de sus transacciones, SQL permite la especificación de tres características de una transacción: el modo de acceso, el tamaño de los diagnósticos y el nivel de aislamiento. El **tamaño de los diagnósticos** determina el número de condiciones de error que pueden registrarse; no se comentará esta característica en más profundidad.

Si el **modo de acceso** es **READ ONLY**, no se permite que la transacción modifique la base de datos. De este modo, no se pueden ejecutar comandos **INSERT**, **DELETE**, **UPDATE**, ni **CREATE**. Si se tiene que ejecutar uno de estos comandos, el modo de acceso debe ser **READ WRITE**. Las transacciones con modo de acceso **READ ONLY** sólo necesitan obtener bloqueos compartidos, incrementando de este modo la concurrencia.

El **nivel de aislamiento** controla el nivel al que se expone la transacción a las acciones de otras transacciones concurrentes. Eligiendo entre cuatro valores de aislamiento posibles, el usuario puede obtener mayor concurrencia con el coste de incrementar la exposición de la transacción a los cambios no comprometidos de otras transacciones.

Los valores del nivel de aislamiento son **READ UNCOMMITTED**, **READ COMMITTED**, **REPEATABLE READ** y **SERIALIZABLE**. El efecto de estos niveles está resumido en la Figura 8.10. En este contexto, la *lectura desfasada* y la *lectura no repetible* se definen como se ha indicado anteriormente.

Nivel	Lectura desfasada	Lectura no repetible	Fantasma
READ UNCOMMITTED	Es posible	Es posible	Es posible
READ COMMITTED	No	Es posible	Es posible
REPEATABLE READ	No	No	Es posible
SERIALIZABLE	No	No	No

Figura 8.10 Niveles de aislamiento de transacciones en SQL-92

El mayor grado de aislamiento respecto a los efectos de otras transacciones se consigue seleccionando para una transacción *T* el nivel **SERIALIZABLE**. Este nivel de aislamiento ase-

gura que T lee solamente los cambios hechos por transacciones comprometidas, que ningún valor leído o escrito por T es modificado por ninguna otra transacción hasta que T finalice, y que si T lee un conjunto de valores basado en alguna condición de búsqueda, este conjunto no es modificado por otras transacciones hasta que T finalice (es decir, T evita el fenómeno fantasma).

En términos de una implementación basada en bloqueos, una transacción **SERIALIZABLE** obtiene los bloqueos antes de leer o escribir objetos, incluyendo bloqueos sobre conjuntos de objetos que se requiere que no cambien y los mantiene hasta el final, de acuerdo a B2F estricto.

REPEATABLE READ garantiza que T sólo lee los cambios hechos por transacciones comprometidas y ningún valor leído o escrito por T es modificado por otras transacciones hasta que T finalice. No obstante, T podría experimentar el fenómeno fantasma; por ejemplo, mientras que T examina todos los registros de Marineros con *categoría*=1, otra transacción podría añadir un nuevo registro a Marineros que verifique esta condición, que no sería tenido en cuenta por T .

Una transacción **REPEATABLE READ** asigna los mismos bloqueos que una **SERIALIZABLE**, excepto que no bloquea los índices; es decir, bloquea solamente objetos individuales, no conjuntos de objetos.

READ COMMITTED asegura que T sólo lee los cambios hechos por transacciones comprometidas, y que ningún valor escrito por T es modificado por ninguna transacción hasta que T finalice. Sin embargo, un valor leído por T puede ser modificado por otra transacción mientras T todavía está ejecutándose, y T está expuesto al problema fantasma.

Una transacción **READ COMMITTED** obtiene bloqueos exclusivos antes de escribir objetos y los mantiene hasta el final. También obtiene bloqueos compartidos antes de leer objetos, pero estos bloqueos se liberan inmediatamente; su único efecto es garantizar que la transacción que modificó el objeto por última vez haya finalizado. (Esta garantía se basa en el hecho de que *toda* transacción SQL obtiene bloqueos exclusivo antes de escribir objetos y los mantiene hasta el final.)

Una transacción **READ UNCOMMITTED** T puede leer los cambios hechos a un objeto por una transacción en curso; obviamente, el objeto puede modificarse posteriormente mientras T esté ejecutándose, y T también es vulnerable al problema fantasma.

Una transacción **READ UNCOMMITTED** no obtiene bloqueos compartidos antes de leer objetos. Este modo representa el mayor nivel de exposición a cambios no comprometidos de otras transacciones; tanto es así que SQL prohíbe que este tipo de transacciones hagan ningún cambio —las transacciones **READ UNCOMMITTED** deben tener modo de acceso **READ ONLY**. Como este tipo de transacciones no obtienen bloqueos de lectura y no está permitido que escriban objetos (y por tanto nunca solicitan bloqueos exclusivos), nunca hacen una petición de bloqueo.

El nivel de aislamiento **SERIALIZABLE** generalmente es el más seguro y es el recomendado para la mayor parte de las transacciones. Algunas transacciones, sin embargo, pueden ejecutarse con un nivel de aislamiento menor, y el menor número de bloqueos solicitado puede contribuir a mejorar el rendimiento del sistema. Por ejemplo, una consulta estadística que obtenga la edad promedio de los marineros puede ejecutarse en el nivel **READ COMMITTED** o incluso en el nivel **READ UNCOMMITTED**, pues unos cuantos valores incorrectos o perdidos no afectan significativamente al resultado si el número de marineros es grande.

El nivel de aislamiento y el modo de acceso puede asignarse mediante el comando **SET TRANSACTION**. Por ejemplo, con el siguiente comando se declara que la transacción actual sea **SERIALIZABLE** y **READ ONLY**:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ ONLY
```

Cuando se inicia una transacción, los valores por defecto son **SERIALIZABLE** y **READ WRITE**.

8.7 PREGUNTAS DE REPASO

Las respuestas a las preguntas de repaso se pueden encontrar en los apartados indicados.

- ¿Qué son las propiedades ACID? Defínanse *atomicidad*, *consistencia*, *aislamiento* y *durabilidad* e ilústrense con ejemplos. (**Apartado 8.1**)
- Defínanse los términos *transacción*, *planificación*, *planificación completa* y *planificación secuencial*. (**Apartado 8.2**)
- ¿Por qué un SGBD entrelaza transacciones concurrentes? (**Apartado 8.3**)
- ¿Cuándo dos acciones sobre el mismo objeto de datos *entrar en conflicto*? Defínanse las anomalías que pueden estar causadas por acciones conflictivas (*lecturas desfasadas*, *lecturas no repetibles*, *actualizaciones perdidas*). (**Apartado 8.3**)
- ¿Qué es una *planificación secuenciable*? ¿Qué es una *planificación recuperable*? ¿Qué es una planificación que *evite abortos en cascada*? ¿Qué es una *planificación estricta*? (**Apartado 8.3**)
- ¿Qué es un *protocolo de bloqueo*? Describase el protocolo de *dos fases estricto (B2F estricto)*. ¿Qué puede decirse de las planificaciones permitidas por este protocolo? (**Apartado 8.4**)
- ¿Qué sobrecargas están asociadas con el control de concurrencia basado en bloqueos? Analíicense específicamente las sobrecargas de *bloquear* y de *abortar* transacciones y explíquese cuál es más importante en la práctica. (**Apartado 8.5**)
- ¿Qué es la pugna? ¿Qué debería hacer un DBA si el sistema entra en pugna? (**Apartado 8.5**)
- ¿Cómo puede incrementarse la productividad? (**Apartado 8.5**)
- ¿Cómo se crean y se finalizan las transacciones en SQL? ¿Qué son los puntos de almacenamiento? ¿Qué son las transacciones encadenadas? Explíquese por qué son útiles los puntos de almacenamiento y las transacciones encadenadas. (**Apartado 8.6**)
- ¿Cuáles son las consideraciones a tener en cuenta al determinar la granularidad del bloqueo cuando se ejecutan sentencias SQL? ¿Qué es el problema fantasma? ¿Qué impacto tiene en el rendimiento? (**Apartado 8.6.2**)

- ¿Qué características de las transacciones puede controlar un programador en SQL? Analíicense en particular los diferentes *modos de acceso y niveles de aislamiento*. ¿Qué problemas deben considerarse al seleccionar un modo de acceso y un nivel de aislamiento para una transacción? (**Apartado 8.6.3**)
- Describábase la forma en que se implementan los distintos niveles de aislamiento en términos de los bloqueos que se asignan. ¿Qué puede decirse de las sobrecargas de bloqueo correspondientes? (**Apartado 8.6.3**)

EJERCICIOS

Ejercicio 8.1 Respóndase brevemente a las siguientes preguntas:

1. ¿Qué es una transacción? ¿En qué es diferente de un programa ordinario (en un lenguaje como C)?
2. Defínanse los siguientes términos: *atomicidad, consistencia, aislamiento, durabilidad, planificación, escritura a ciegas, lectura sucia, lectura no repetible, planificación secuenciable, planificación recuperable, planificación que evita abortos en cascada*.
3. Describábase el protocolo B2F estricto.
4. ¿Qué es el problema fantasma? ¿Puede ocurrir en una base de datos en la que el conjunto de objetos de base de datos es fijo y sólo los valores de los objetos pueden cambiar?

Ejercicio 8.2 Considérense las siguientes acciones tomadas por la transacción T_1 sobre los objetos de base de datos X e Y :

$L(X), E(X), L(Y), E(Y)$

1. Proporcióñese un ejemplo de otra transacción T_2 que, si se ejecuta concurrentemente con T_1 sin ninguna forma de control de concurrencia, podría interferir con T_1 .
2. Explíquese la forma en que el uso de B2F estricto previene la interferencia entre las dos transacciones.
3. B2F estricto se utiliza en muchos sistemas de base de datos. Proporcionense dos razones de su popularidad.

Ejercicio 8.3 Considérese una base de datos con objetos X e Y y supóngase que hay dos transacciones T_1 y T_2 . La transacción T_1 lee los objetos X e Y y después escribe el objeto X . La transacción T_2 lee los objetos X e Y y después escribe los objetos X e Y .

1. Proporcióñese una planificación de ejemplo con las acciones de las transacciones T_1 y T_2 sobre los objetos X e Y que resulte en un conflicto de escritura-lectura.
2. Proporcióñese una planificación de ejemplo con las acciones de las transacciones T_1 y T_2 sobre los objetos X e Y que resulte en un conflicto de lectura-escritura.
3. Proporcióñese una planificación de ejemplo con las acciones de las transacciones T_1 y T_2 sobre los objetos X e Y que resulte en un conflicto de escritura-escritura.
4. Para cada una de las tres planificaciones, demuéstrese que el protocolo B2F estricto no las permitiría.

Ejercicio 8.4 Una transacción que solamente lee objetos de la base de datos se denomina una transacción de **sólo lectura**, en caso contrario se dice que es de **lectura-escritura**. Respóndase brevemente a las siguientes preguntas:

1. ¿Qué es la pugna de transacciones y cuándo ocurre?
2. ¿Qué le ocurre a la productividad del sistema de base de datos si el número de transacciones de lectura-escritura se incrementa?
3. ¿Qué le ocurre a la productividad del sistema de base de datos si el número de transacciones de sólo lectura se incrementa?

4. Describanse tres formas de ajustar el sistema para incrementar la productividad de las transacciones.

Ejercicio 8.5 Supóngase que un SGBD reconoce como acciones *incrementar*, que incrementa en 1 un objeto que contiene un valor entero, y *decrementar*, además de las lecturas y las escrituras. Una transacción que incrementa un objeto no necesita saber el valor del objeto; incrementar y decrementar son versiones de escrituras ciegas. Además de los bloqueos compartidos y exclusivos, se permiten dos bloqueos especiales: un objeto debe bloquearse en modo *I* antes de incrementarlo, y en modo *D* antes de decrementarlo. Un bloqueo *I* es compatible con otro bloqueo *I* o *D* sobre el mismo objeto, pero no es compatible con los bloqueos *C* y *X*.

1. Ilústrese la forma en que el uso de los bloqueos *I* y *D* puede incrementar la concurrencia. (Muéstrese una planificación permitida por el protocolo B2F estricto que sólo use bloqueos *C* y *X*. Explíquese la forma en que el uso de los bloqueos *I* y *D* puede permitir más acciones entrelazadas, mientras continúa siguiendo el protocolo B2F estricto.)
2. Explíquese informalmente la manera en que el protocolo B2F estricto garantiza la secuencialidad aun en presencia de bloqueos *I* y *D*. (Identifíquense los pares de acciones que están en conflicto, y muéstrese que el uso de los bloqueos *C*, *X*, *I* y *D* de acuerdo con B2F estricto ordena todos los pares de acciones conflictivas de igual forma que alguna planificación en serie.)

Ejercicio 8.6 SQL soporta cuatro niveles de aislamiento y dos modos de acceso, con un total de ocho combinaciones de ambos. Cada combinación define implícitamente una clase de transacciones; las siguientes preguntas se refieren a estas ocho clases:

1. Considérense los cuatro niveles de aislamiento. Describase cuál de los siguientes fenómenos puede ocurrir en cada uno de estos niveles: *lectura sucia*, *lectura no repetible*, *problema fantasma*.
2. Para cada uno de los cuatro niveles de aislamiento, proporcionense ejemplos de transacciones que puedan ejecutarse de forma segura en ese nivel.
3. ¿Por qué es importante el modo de acceso de una transacción?

Ejercicio 8.7 Considérese el esquema de base de datos de matrícula universitaria:

```
Alumnos(nume: integer, nombree: string, carrera: string, nivel: string, edad: integer)
Clases(nombrec: string, horario: time, aula: string, idp: integer)
Matriculado(nume: integer, nombrec: string)
Profesores(idp: integer, nombrep: string, iddepto: integer)
```

El significado de estas relaciones es directo; por ejemplo, Matriculado tiene un registro por par alumno-clase tal que el alumno está matriculado en la clase.

Para cada una de las siguientes transacciones, dígase el nivel de aislamiento SQL que se utilizaría y explíquese por qué.

1. Matrícula de un alumno identificado por *nume* en la clase denominada “Introducción a los sistemas de base de datos”.
2. Cambio de la matrícula de un alumno identificado por *nume* de una clase a otra.
3. Asignación de un nuevo profesor identificado por *idp* a la clase con el menor número de alumnos.
4. Por cada clase, listado del número de alumnos inscritos.

Ejercicio 8.8 Considérese el siguiente esquema:

```
Proveedores(idp: integer, nombrep: string, dirección: string)
Artículos(ida: integer, nombrea: string, color: string)
Catálogo(idp: integer, ida: integer, coste: real)
```

La relación Catálogo lista los precios de los Artículos suministrados por los Proveedores.

Por cada una de las siguientes transacciones, dígase el nivel de aislamiento SQL que se debe utilizar y explíquese por qué.

1. Una transacción que añada un nuevo artículo al catálogo de un proveedor.
2. Una transacción que incremente el precio de un artículo suministrado por un proveedor.
3. Una transacción que determine el número total de artículos de un proveedor determinado.
4. Una transacción que muestre, por cada artículo, el proveedor que lo suministra al precio más barato.

Ejercicio 8.9 Considérese una base de datos con el siguiente esquema:

```
Proveedores(idp: integer, nombrep: string, dirección: string)
Artículos(ida: integer, nombrea: string, color: string)
Catálogo(idp: integer, ida: integer, coste: real)
```

La relación Catálogo lista los precios de los Artículos suministrados por los Proveedores.

Considérense las transacciones T_1 y T_2 . T_1 siempre tiene un nivel de aislamiento SQL **SERIALIZABLE**. Se ejecuta primero T_1 concurrentemente con T_2 y después se ejecutan concurrentemente pero cambiando el nivel de aislamiento de T_2 como se especifica a continuación. Dada un ejemplar de la base de datos y sentencias SQL para T_1 y T_2 tales que el resultado de ejecutar T_2 con el primer nivel de aislamiento sea diferente de ejecutarlo con el segundo nivel de aislamiento. Especifíquese también la planificación común de T_1 y T_2 y explíquese por qué los resultados son diferentes.

1. **SERIALIZABLE** frente a **REPEATABLE READ**.
2. **REPEATABLE READ** frente a **READ COMMITTED**.
3. **READ COMMITTED** frente a **READ UNCOMMITTED**.

NOTAS BIBLIOGRÁFICAS

El concepto de transacción y algunas de sus limitaciones se tratan en [216]. Un modelo de transacciones formal que generaliza varios modelos anteriores se propone en [115].

El bloqueo en dos fases fue introducido en [162], un artículo fundamental que también analiza los conceptos de transacciones, fantasmas y bloqueos de predicado. El tratamiento formal de la serializabilidad aparece en [55, 351].

Una excelente presentación en profundidad del procesamiento de transacciones puede encontrarse en [54] y [468]. [219] es un clásico, con un tratamiento enciclopédico de este tema.

PARTE III

ALMACENAMIENTO E ÍNDICES



9

INTRODUCCIÓN AL ALMACENAMIENTO Y LOS ÍNDICES

- ¿Cómo almacena y accede a datos persistentes un SGBD?
- ¿Por qué el coste de E/S es tan importante para las operaciones de base de datos?
- ¿Cómo organiza un SGBD los archivos de registros de datos en disco para minimizar el coste de E/S?
- ¿Qué es un índice y por qué se usa?
- ¿Cuál es la relación entre un archivo de registros de datos y cualquiera de los índices sobre este archivo?
- ¿Qué propiedades importantes tienen los índices?
- ¿Cómo funciona un índice asociativo y cuándo es más efectivo?
- ¿Cómo funciona un índice basado en árboles y cuándo es más efectivo?
- ¿Cómo se pueden utilizar índices para optimizar el rendimiento para una carga de trabajo dada?
- **Conceptos clave:** almacenamiento externo, gestor de la memoria intermedia, E/S de páginas; organización de archivos, archivos de montículo, archivos ordenados; índices, entradas de datos, claves de búsqueda, índice agrupado, archivo agrupado, índice primario; organización de índices, índices asociativos y basados en árboles; comparación de coste, organización y operaciones comunes de archivos; ajuste del rendimiento, carga de trabajo, claves de búsqueda compuestas, uso del agrupamiento.

Si no lo encuentra en el índice, revise muy cuidadosamente todo el catálogo.

— Sears, Roebuck, and Co., Guía del consumidor, 1897

La abstracción básica de los datos en un SGBD es una colección de registros, o *archivo*, y cada archivo está compuesto de una o más páginas. La capa de software de *archivos y métodos de acceso* organiza los datos cuidadosamente para permitir un acceso rápido a los subconjuntos de registros deseados. Entender cómo están organizados los registros es esencial para utilizar un sistema de bases de datos de forma efectiva, y es el tema principal de este capítulo.

Una **organización de archivo** es un método de organizar los registros en un archivo cuando se almacena en disco. Cada organización de archivo hace determinadas operaciones eficientes pero otras costosas.

Considérese un archivo de registros de empleados, cada uno de ellos con los campos *edad*, *nombre* y *sueldo*, que utilizaremos como ejemplo en este capítulo. Si se desea recuperar los registros de empleados en orden creciente de edad, una buena organización del archivo es ordenarlo de esa forma, pero este orden es de mantenimiento costoso si el archivo se modifica frecuentemente. Más aún, frecuentemente resulta interesante permitir más de una operación sobre una colección de registros dada. En el ejemplo también se pueden determinar todos los empleados que ganen más de 5.000 €. Sería necesario recorrer todo el archivo para encontrar los registros.

La técnica denominada *indexación* puede ayudar cuando hay que acceder a una colección de registros de múltiples formas, además de permitir eficientemente varias clases de selección. El Apartado 9.2 introduce la indexación, un aspecto importante de la organización de archivos en un SGBD. En el Apartado 9.3 se presenta una visión general de las estructuras de datos de indexación; en los Capítulos 10 y 11 se incluye un tratamiento más detallado.

La importancia de elegir la organización de archivo apropiada se ilustra en el Apartado 9.4 mediante un análisis simplificado de varias organizaciones de archivos alternativas. El modelo de coste utilizado en este análisis, presentado en el Apartado 9.4.1, se utiliza también en capítulos posteriores. En el Apartado 9.5 se resaltan algunas decisiones importantes a tomar cuando se crean índices. Podría decirse que elegir una buena colección de índices a construir es la herramienta más poderosa que un administrador de base de datos tiene para mejorar el rendimiento.

9.1 DATOS EN ALMACENAMIENTO EXTERNO

Un SGDB almacena enormes cantidades de datos, y éstos deben persistir entre distintas ejecuciones del programa. Por tanto, los datos se almacenan en dispositivos de almacenamiento externo tales como discos y cintas, y se traen a memoria principal cuando se necesiten para algún procesamiento. La unidad de información leída de o escrita en disco es una *página*. El tamaño de una página es un parámetro del SGBD, los valores típicos son 4KB o 8KB.

El coste de E/S de páginas (*entrada* de disco a memoria principal y *salida* de memoria a disco) domina el coste de las operaciones típicas de bases de datos, y los sistemas de bases de datos están optimizados cuidadosamente para minimizarlo. Omitiendo los detalles del modo en que se almacenan físicamente los archivos de registros en el disco y el uso de la memoria principal, es importante tener en cuenta los siguientes puntos:

- Los discos son los dispositivos de almacenamiento externo más importantes. Permiten recuperar cualquier página con un coste (más o menos) fijo por página. Sin embargo, si

se leen varias páginas en el orden en el que están físicamente almacenadas, el coste puede ser mucho menor que el de leer las mismas páginas en un orden aleatorio.

- Las cintas son dispositivos de acceso secuencial que fuerzan a leer una página detrás de otra. Se utilizan principalmente para archivar datos que no son necesarios frecuentemente.
- Cada registro de un archivo tiene un identificador único denominado **identificador de registro**, o **idr** para abreviar. Los idr tienen la propiedad de que permiten identificar la dirección en disco de la página que contiene el registro asociado.

Los datos son cargados en memoria para ser procesados, y escritos en disco para almacenarlos persistentemente, por una capa de software llamada *gestor de la memoria intermedia*. Cuando la capa de *archivos y métodos de acceso* (también conocida como la capa de archivos) necesita procesar una página, pide al gestor de la memoria intermedia que la recupere, especificando el idr de la página. El gestor de la memoria intermedia trae la página de disco si no está ya en memoria.

El *gestor de espacio en disco* gestiona el espacio libre y ocupado en disco según la arquitectura software del SGBD descrita en el Apartado 1.8. Cuando la capa de archivos y métodos de acceso necesita espacio adicional para dejar nuevos registros en un archivo, pide al gestor de espacio en disco que asigne una página adicional de disco al archivo; también informa al gestor de espacio en disco cuando ya no necesita más una de sus páginas. El gestor de espacio en disco lleva un registro de las páginas que están en uso por la capa de archivos; si una página es liberada por la capa de archivos, el gestor de espacio reutiliza el espacio si la capa de archivos solicita una página nueva posteriormente.

El resto del capítulo se centra en la capa de archivos y métodos de acceso.

9.2 ORGANIZACIONES DE ARCHIVO E INDEXACIÓN

El **archivo de registros** es una abstracción importante de un SGBD, y está implementada por la capa de archivos y métodos de acceso. Un archivo puede crearse, destruirse y se le pueden insertar y borrar registros. También soporta exploraciones; la operación de **exploración** permite recorrer todos los registros del archivo de uno en uno. Una relación normalmente está almacenada como un archivo de registros.

La capa de archivos almacena los registros de un archivo en una colección de páginas de disco. Lleva la cuenta de las páginas dedicadas a cada archivo y, como los registros se insertan y eliminan del archivo, también mantiene un registro del espacio disponible en las páginas asignadas al archivo.

La estructura de archivo más simple es un archivo no ordenado o **archivo de montículo**. Los registros de un archivo de montículo se almacenan en orden aleatorio en las páginas del archivo. Esta organización soporta la recuperación de todos los registros, o la de un registro determinado especificado por su idr; el gestor de archivos debe llevar registro de las páginas asignadas al archivo.

Un **índice** es una estructura de datos que organiza los registros en disco para optimizar cierto tipo de operaciones de lectura. Un índice permite recuperar eficientemente todos los registros que satisfacen condiciones de búsqueda sobre los campos **clave de búsqueda** del índice. También pueden crearse índices adicionales sobre una colección de registros de da-

tos dada, cada uno con una clave de búsqueda diferente, para acelerar las operaciones de búsqueda que no están soportadas eficientemente por la organización de archivo utilizada para almacenar los registros.

Considérese el ejemplo de registros de empleados. Es posible almacenar los registros en un archivo organizado como un índice sobre la edad del empleado; ésta es una alternativa a la ordenación del archivo por edad. Además, se puede crear un archivo índice auxiliar basado en el sueldo para acelerar las consultas en las que intervenga. El primer archivo contiene registros de empleados, y el segundo contiene registros que permiten localizar los registros de empleados que satisfacen una condición sobre el sueldo.

Se utiliza el término **entrada de datos** para referirse a los registros almacenados en un archivo de índice. Una entrada de datos con valor de clave de búsqueda k , denotado como $k*$, contiene suficiente información para localizar (uno o más) registros de datos con el valor de clave k . Es posible hacer búsquedas eficientes sobre un índice para encontrar las entradas de datos deseadas, y después utilizarlas para obtener sus registros de datos (si éstos son distintos de las entradas de datos).

Existen tres alternativas principales sobre lo que se debe almacenar en una entrada de datos de un índice:

1. Una entrada de datos $k*$ es un registro de datos real (con valor de clave de búsqueda k).
2. Una entrada de datos es un par $\langle k, idr \rangle$, donde idr es el identificador de registro de datos con valor de clave k .
3. Una entrada de datos es un par $\langle k, lista-idr \rangle$, donde $lista-idr$ es una lista de identificadores de registros de datos con valor de clave k .

Por supuesto, si se utiliza el índice para almacenar los registros de datos reales, alternativa (1), cada entrada $k*$ es un registro de datos con valor de clave k . Se puede pensar en este índice como una organización de archivo especial. Esta **organización de archivo indexado** se puede utilizar en lugar de, por ejemplo, un archivo de registros ordenado o uno no ordenado.

Las alternativas (2) y (3), que contienen entradas de datos que apuntan a registros de datos, son independientes de la organización de archivo que se utilice para el archivo indexado (es decir, el archivo que contiene los registros de datos). La alternativa (3) ofrece una mejor utilización del espacio que la alternativa (2), pero las entradas de datos son de longitud variable, dependiendo del número de registros de datos con un valor dado de clave.

Si se quiere crear más de un índice sobre una colección de registros de datos —por ejemplo, se quieren crear índices tanto sobre el campo *edad* como sobre *sueldo* en la colección de registros Empleados— solamente uno de los índices debe utilizar la alternativa (1), pues se debe evitar almacenar registros de datos varias veces.

9.2.1 Índices agrupados

Cuando un archivo está organizado de forma que el orden de los registros de datos es el mismo o parecido al orden de las entradas de datos de algún índice, se dice que el índice está **agrupado**; en caso contrario, es un índice **no agrupado**. Un índice que utiliza la alternativa (1) está agrupado por definición. Un índice que utiliza la alternativa (2) o (3) puede ser un índice agrupado sólo si los registros de datos están ordenados por el campo

clave de búsqueda. De lo contrario, el orden de los registros de datos es aleatorio, definido exclusivamente por su orden físico, y no hay forma razonable de organizar las entradas de datos del índice en el mismo orden.

En la práctica, los archivos raramente están ordenados porque es demasiado caro mantener el orden cuando se actualizan los datos. Así que, en la práctica, un índice agrupado es un índice que utiliza la alternativa (1), y los índices que utilizan las alternativas (2) o (3) son no agrupados. A veces se usará el término **archivo agrupado** para denotar a un índice bajo la alternativa (1), porque las entradas de datos son registros reales de datos, y por tanto el índice es un archivo de registros de datos. (Como se ha observado antes, las búsquedas y exploraciones sobre un índice devuelven solamente sus entradas de datos, incluso si contienen información adicional para organizarlas.)

El coste de utilizar un índice para responder a una consulta de búsqueda por rango puede variar enormemente si el índice está agrupado o no. Si lo está, es decir, se está usando la clave de búsqueda de un archivo agrupado, los idr que determinan las entradas de datos apuntan a una colección contigua de registros, y solamente es necesario recuperar algunas páginas. Si el índice no es agrupado, cada entrada de datos del rango puede contener un idr que apunte a una página de datos distinta, lo que acaba en tantas E/S de páginas de datos como el número de entradas de datos que corresponden al rango, como se ilustra en la Figura 9.1.

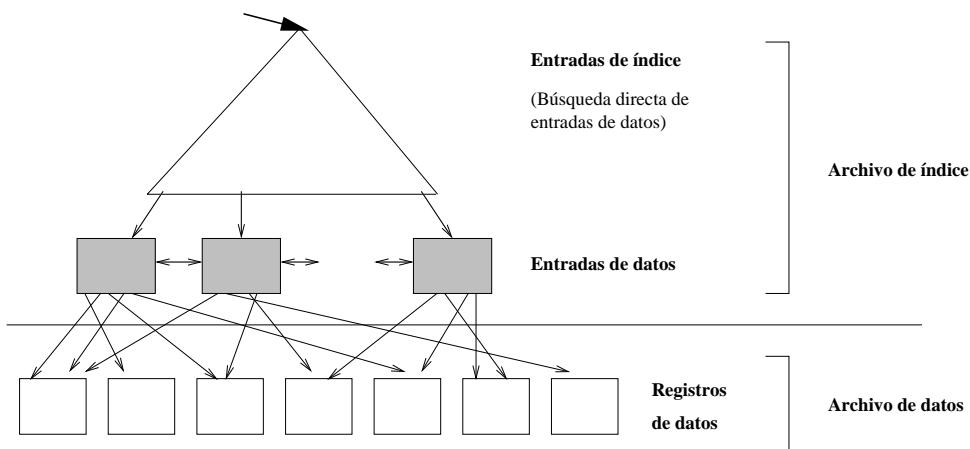


Figura 9.1 Índice no agrupado bajo la alternativa (2)

9.2.2 Índices primario y secundario

Un índice sobre un conjunto de campos que incluye la *clave primaria* (véase el Capítulo 3) se denomina **índice primario**; el resto de índices se denominan **índices secundarios**. (Los términos *índice primario* e *índice secundario* a veces se utilizan con un significado diferente: un índice que utiliza la alternativa (1) se denomina *primario*, y si utilizan las alternativas (2) o (3) se denomina *secundario*. En este libro se seguirán las definiciones presentadas anteriormente, pero el lector debería ser consciente de esta falta de terminología estándar en la literatura.)

Se dice que dos entradas de datos están **duplicadas** si tienen el mismo valor del campo clave de búsqueda asociado con el índice. Un índice primario garantiza que no contiene duplicados, pero un índice sobre otros (conjuntos de) campos puede contener duplicados. En general, un índice secundario contiene duplicados. Si se sabe que no existen duplicados, se dice que es un índice **único**.

Un problema importante es cómo se organizan las entradas de datos de un índice para permitir la recuperación eficiente de entradas, lo cual se trata a continuación.

9.3 ESTRUCTURAS DE DATOS DE ÍNDICES

Una forma de organizar las entradas de datos es asociar las entradas de datos a partir de la clave de búsqueda. Otra forma es construir una estructura de datos en forma de árbol que dirija la búsqueda de entradas de datos. La indexación basada en árboles se tratará con más detalle en el Capítulo 10 y la indexación asociativa en el Capítulo 11.

Obsérvese que la elección de la técnica de indexación asociativa o de árbol puede combinarse con cualquiera de las tres alternativas de entradas de datos.

9.3.1 Indexación asociativa (*Hash*)

Se pueden organizar registros utilizando una técnica denominada *asociación (hashing)* para encontrar rápidamente los registros que tienen un valor determinado de la clave de búsqueda. Por ejemplo, si el archivo de registros de empleados está asociado sobre el campo *nombre*, es posible recuperar todos los registros sobre Juan.

En este enfoque los registros de un archivo se agrupan en **cajones** (*buckets*), donde un cajón consiste en una **página primaria** y, posiblemente, páginas adicionales enlazadas en una cadena. Puede determinarse el cajón al que pertenece un registro aplicando una función especial, denominada **función de asociación** (*hash*), a la clave de búsqueda. Dado un número de cajón, una estructura de índice asociativa permite recuperar la página primaria del cajón en uno o dos accesos E/S en disco.

En las inserciones el registro se introduce en el cajón apropiado y asignando páginas de desbordamiento si fuera necesario. Para buscar un registro con un valor de clave dado se aplica la función de asociación para identificar el cajón al que pertenece el registro y mirar en todas las páginas del cajón. Si el valor de clave del registro no está disponible, por ejemplo cuando el índice está basado en *sueldo* pero se necesitan extraer los registros con un valor determinado de *edad*, es necesario explorar todas las páginas del archivo.

En este capítulo se asumirá que la aplicación de la función de asociación a (la clave de búsqueda de) un registro permite identificar y recuperar la página que lo contiene con una sola operación de E/S. En la práctica se conocen estructuras de índices asociativos que se ajustan con elegancia a las inserciones y borrados y que permiten recuperar la página que contiene un registro en una o dos operaciones de E/S (véase el Capítulo 11).

La indexación por asociación se muestra en la Figura 9.2, donde los datos se almacenan en un archivo que está asociado por *edad*; las entradas de datos en este primer archivo de índices son los registros de datos. Aplicando la función de asociación al campo *edad* se identifica la página a la que pertenece el registro. La función de asociación h de este ejemplo es bastante

simple; convierte el valor de la clave a su representación binaria y usa los dos bits menos significativos como identificador de cajón.

La Figura 9.2 también muestra un índice con clave de búsqueda *sueldo* que contiene pares $\langle \text{sueldo}, \text{idr} \rangle$ como entradas de datos. El componente *idr* (abreviatura de *identificador de registro*) de las entradas de datos en este segundo índice es un puntero a un registro con valor de clave *sueldo* (que se muestra en la figura como una flecha que apunta al registro de datos).

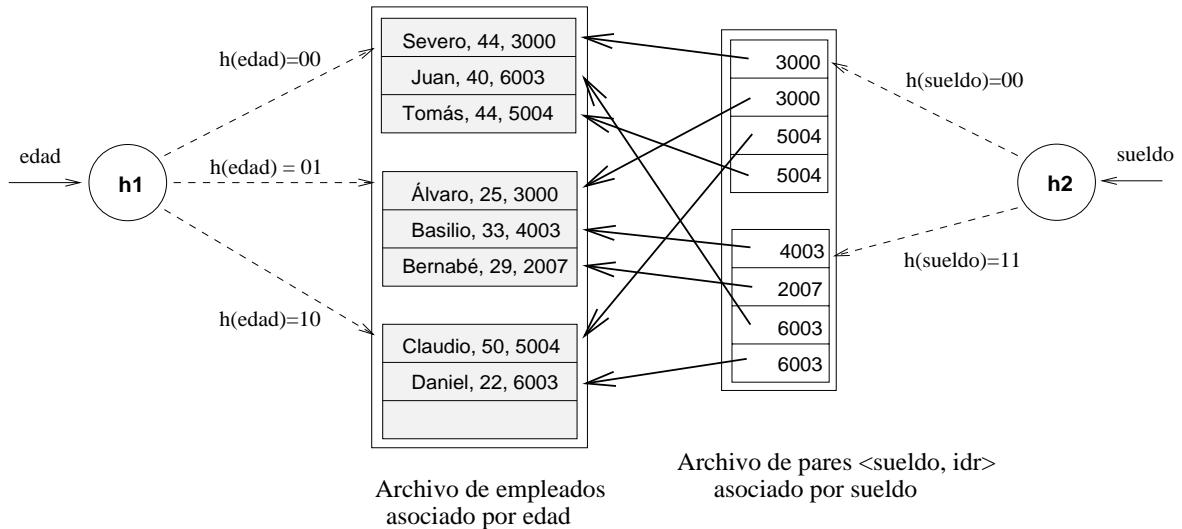


Figura 9.2 Archivo organizado por índice asociativo respecto a *edad*, con índice auxiliar por *sueldo*

Utilizando la terminología introducida en el Apartado 9.2, la Figura 9.2 ilustra las alternativas (1) y (2) para las entradas de datos. El archivo de registros de empleados es asociativo por *edad*, y se utiliza la alternativa (1) para las entradas de datos. El segundo índice, por *sueldo*, también utiliza asociación para localizar entradas de datos, que ahora son pares $\langle \text{sueldo}, \text{idr} \rangle$ de *registro de empleados*; es decir, la alternativa (2) se utiliza para las entradas de datos.

Nótese que la clave de búsqueda de un índice puede ser cualquier secuencia de uno o más campos, y no necesitan identificar los registros únicamente. Por ejemplo, en el índice de sueldo, dos entradas de datos tienen el mismo valor de clave 6003. (Existe una sobrecarga desafortunada del término *clave* en la literatura de bases de datos. Una *clave primaria* o *clave candidata* —campos que identifican de forma única un registro; véase el Capítulo 3— no tiene que ver con el concepto de clave de búsqueda.)

9.3.2 Indexación basada en árboles

Una alternativa a la indexación asociativa es la organización de los registros utilizando una estructura de datos en forma de árbol. Las entradas de datos se ordenan por valor de clave de búsqueda, y se mantiene una estructura de datos jerárquica que dirige las búsquedas a la página de entradas de datos correcta.

La Figura 9.3 muestra los registros de empleados de la Figura 9.2, esta vez organizados en un índice estructurado en árbol con clave de búsqueda *edad*. Cada nodo en esta figura

(por ejemplo, los nodos etiquetados A, B, H1, H2) es una página física, y recuperar un nodo implica una operación de E/S en disco.

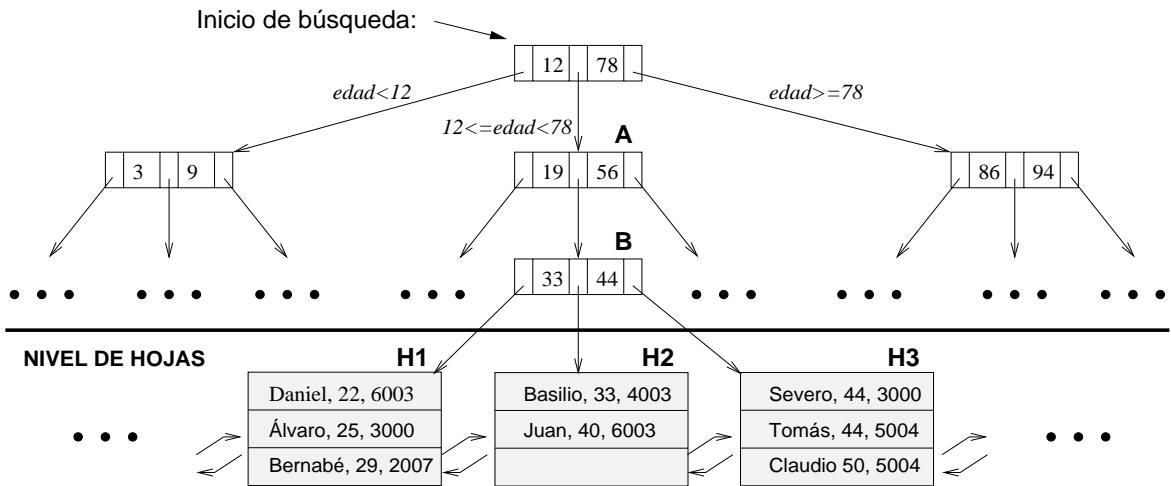


Figura 9.3 Índice estructurado en árbol

El nivel más bajo del árbol, llamado **nivel de hojas**, contiene las entradas de datos; en el ejemplo, éstos son registros de empleados. Para ilustrar esto mejor, la Figura 9.3 está dibujada como si hubiera registros adicionales de empleados, algunos con edad inferior a 22 y otros con edad superior a 50 (los valores mínimo y máximo que aparecen en la Figura 9.2). Los registros adicionales con edad inferior a 22 aparecerían en páginas hoja a la izquierda de la página H1, y los registros con edad superior a 50 aparecerían en páginas hoja a la derecha de la página H3.

Esta estructura permite localizar eficientemente todas las entradas de datos con valores de clave en un rango determinado. Todas las búsquedas comienzan en el nodo más alto, denominado **raíz**, y los contenidos de las páginas en niveles que no son hojas dirigen las búsquedas a las páginas hoja correctas. Las páginas que no son hojas contienen punteros a nodos separados por valores de clave de búsqueda. El puntero de nodo a la izquierda de un valor de clave k apunta a un subárbol que contiene solamente entradas de datos menores que k . El puntero de nodo a la derecha de un valor de clave k apunta a un subárbol que contiene solamente entradas de datos mayores o iguales que k .

Supóngase que se desean determinar todas las entradas de datos del ejemplo tales que $24 < edad < 50$. Cada arco desde el nodo raíz a un nodo hijo en la Figura 9.2 tiene una etiqueta que explica lo que contiene el subárbol correspondiente. (Aunque en la figura no se muestran las etiquetas de los arcos restantes, deberían ser fáciles de deducir.) En la búsqueda de ejemplo se buscan entradas de datos con valor de clave > 24 , llegando al nodo de en medio, el nodo A. Examinando de nuevo el contenido de este nodo, somos dirigidos al nodo B. Examinando su contenido, se pasa al nodo hoja H1, que contiene las entradas de datos buscadas.

Obsérvese que los nodos hoja H2 y H3 también contienen entradas de datos que satisfacen el criterio de búsqueda. Para facilitar la recuperación de estas entradas durante la búsqueda

se mantienen todas las páginas hoja en una lista doblemente enlazada. Así es posible leer la página H2 utilizando el puntero “siguiente” de la página H1, y después leer la página H3 mediante el puntero de la página H2.

De este modo, el número de operaciones de E/S realizadas durante una búsqueda es igual a la longitud del camino desde la raíz a una hoja, más el número de páginas hoja con entradas de datos que satisfacen la condición de búsqueda. Los **árboles B+** son una estructura de índice que asegura que todos los caminos desde la raíz a las hojas en un árbol dado son de la misma longitud, es decir, la estructura siempre está equilibrada (*balanced*) en altura. Buscar la página hoja correcta es más rápido que la búsqueda binaria de las páginas en un archivo ordenado porque cada nodo que no es hoja puede alojar un gran número de punteros a nodos, y la altura del árbol raramente es más de tres o cuatro en la práctica. La **altura** de un árbol equilibrado es la longitud de un camino de la raíz a las hojas; en la Figura 9.3 la altura es tres. El número de operaciones de E/S para recuperar una página hoja es cuatro, incluyendo la raíz y la página hoja. (En la práctica, la raíz está normalmente en la memoria intermedia porque se accede a ella frecuentemente, y realmente se producen sólo tres operaciones de E/S en un árbol de altura tres.)

El número medio de hijos de un nodo que no es hoja se denomina el **grado de salida** del árbol. Si cada nodo que no es hoja tiene n hijos, un árbol de altura a tendrá n^a páginas hijas. En la práctica, los nodos no tienen el mismo número de hijos, pero utilizando el valor medio F para n , es posible obtener una buena aproximación del número de páginas hoja, F^a . En la práctica F es por lo menos 100, lo que quiere decir que un árbol de altura cuatro contiene 100 millones de páginas hoja. Por tanto, es posible buscar en un archivo con 100 millones de páginas hoja y encontrar la buscada utilizando cuatro operaciones de E/S; una búsqueda binaria del mismo archivo llevaría $\log_2 100.000.000$ (más de 25) operaciones de E/S.

9.4 COMPARACIÓN ENTRE LAS ORGANIZACIONES DE ARCHIVO

En este apartado se comparará el coste de algunas operaciones simples en varias organizaciones de archivo básicas sobre una colección de registros de empleados. Se supondrá que los archivos y los índices están organizados de acuerdo a la clave de búsqueda compuesta (*edad, sueldo*), y que todas las operaciones de selección están especificadas sobre estos campos. Las organizaciones que se considerarán son las siguientes:

- Archivo de registros de empleados ordenados de forma aleatoria, en un archivo de montículo.
- Archivo de registros de empleados ordenado por *(edad, sueldo)*.
- Archivo de árbol B+ agrupado con clave de búsqueda *(edad, sueldo)*.
- Archivo de montículo con un índice de árbol B+ no agrupado por *(edad, sueldo)*.
- Un archivo de montículo con un índice asociativo no agrupado por *(edad, sueldo)*.

El objetivo es resaltar la importancia de la elección de una organización de archivo apropiada, y la lista anterior incluye las alternativas principales a considerar en la práctica. Obviamente, es posible mantener los registros desordenados u ordenarlos. También se puede

elegir construir un índice sobre el archivo de datos. Nótese que incluso si el archivo de datos está ordenado, ¡un índice cuya clave de búsqueda difiera del orden del archivo se comporta como un índice sobre un archivo de montículo!

Las operaciones que se considerarán son:

- **Exploración.** Lee todos los registros del archivo. Las páginas del archivo deben leerse del disco a la memoria intermedia. También hay una sobrecarga de la CPU por registro para localizar el registro en la página (en la memoria intermedia).
- **Búsqueda con selección de igualdad.** Lee todos los registros que satisfacen una selección de igualdad; por ejemplo, “Determinar el registro de empleados con *edad* 23 y *suelto* 50.” Las páginas que contienen los registros seleccionados deben leerse de disco, y éstos deben localizarse dentro de estas páginas.
- **Búsqueda con selección por rango.** Lee todos los registros que satisfacen una selección por rango; por ejemplo, “Determinar todos los registros de empleados con *edad* mayor a 35.”
- **Insertar un registro.** Insertar un registro dado en el archivo. Se debe identificar la página del archivo en la que debe insertarse el registro, leerla página del disco, modificarla para incluir el nuevo registro y escribirla. Dependiendo de la organización del archivo, es posible que también tengamos que leer, modificar y escribir otras páginas.
- **Borrar un registro.** Borrar un registro especificado mediante su idr. Se debe identificar la página que contiene el registro, leerla de disco, modificarla y escribirla en disco. Dependiendo de la organización del archivo, es posible que también sea necesario leer, modificar y escribir otras páginas.

9.4.1 Modelo de coste

En esta comparación de organizaciones de archivo, y en capítulos posteriores, se empleará un modelo de coste sencillo que permite estimar el coste (en términos del tiempo de ejecución) de diferentes operaciones de base de datos. Se usará B para indicar el número de páginas de datos cuando los registros están distribuidos en páginas sin desperdiciar espacio, y R para el número de registros por página. El tiempo medio para leer o escribir una página de disco es D , y el tiempo medio para procesar un registro (por ejemplo, para comparar el valor de un campo con una constante) es C . En la organización de archivo asociativa se utiliza una función, denominada *función de asociación*, para corresponder un registro en un rango de números; el tiempo requerido para aplicar la función de asociación a un registro es H . Para los índices en árbol, se utilizará F para indicar el grado de salida, que normalmente es al menos 100 como se ha comentado en el Apartado 9.3.2.

Los valores típicos actualmente son $D = 15$ milisegundos, $C = 100$ nanosegundos; por tanto es de esperar que domine el coste de E/S. La E/S es a menudo (incluso normalmente) el componente dominante del coste de las operaciones de base de datos, y así considerar el coste de E/S da una buena primera aproximación del coste real. Más aún, La velocidad de la CPU crece de forma constante, mientras que la velocidad de los discos no se incrementa al mismo ritmo. (Por otra parte, según aumentan los tamaños de memoria principal, es probable

que una fracción mayor de las páginas necesarias quepa en memoria, lo que lleva a un menor número de peticiones de E/S.) Por tanto, el modelo de coste se centra en el componente de E/S y se asume que el coste de procesamiento por registro en memoria es la constante C . Es necesario tener presentes las siguientes observaciones:

- Los sistemas reales deben considerar otros aspectos de coste, tales como el coste debido a la CPU (y el coste de transmisión en red en una base de datos distribuida).
- Incluso tomando la decisión de centrarse en el coste de E/S, un modelo preciso sería demasiado complejo para nuestro propósito de transmitir las ideas fundamentales de forma sencilla. Por ello utilizaremos un modelo simple en el que sólo se contará el número de páginas leídas o escritas en disco como medida de E/S. En este análisis se ignorará el tema importante de los **accesos en bloque** —normalmente, los sistemas de disco permiten leer un bloque de páginas contiguas en una sola petición de E/S—. El coste es igual al tiempo requerido para *pedir* la primera página del bloque y transferir todas las páginas del bloque. Estos accesos en bloque pueden ser mucho más económicos que realizar una petición de E/S por página del bloque, especialmente si estas peticiones no se siguen de forma consecutiva, porque aparecería un coste de petición adicional por cada página del bloque.

Las implicaciones del modelo de coste se comentarán siempre que las suposiciones de simplificación puedan afectar a las conclusiones de forma importante.

9.4.2 Archivos de montículo

Exploración. El coste es $B(D + RC)$ porque hay que recuperar cada una de las B páginas, tomando un tiempo D por página, y para cada página, procesar los R registros tomando un tiempo C por registro.

Búsqueda con selección de igualdad. Supóngase que se sabe por adelantado que exactamente un registro satisface la selección de igualdad, es decir, la selección se especifica sobre una clave candidata. En promedio se debe leer la mitad del archivo, suponiendo que el registro existe y que la distribución de los valores en el campo de búsqueda sea uniforme. Para cada página de datos recuperada se deben comprobar todos los registros de la página para ver si está el registro deseado. El coste es $0,5B(D + RC)$. Sin embargo, si no hay ningún registro que satisfaga la selección, debe leerse todo el archivo para comprobarlo.

Si la selección no se hace sobre un campo de clave candidata (por ejemplo, “Determinar los empleados de 18 años”), siempre es necesario explorar todo el archivo porque los registros con $edad = 18$ pueden estar dispersos por todo el archivo, y se desconoce su número.

Búsqueda con selección por rango. Debe explorarse todo el archivo porque los registros seleccionados pueden aparecer en cualquier lugar, y se desconoce su número. El coste es $B(D + RC)$.

Inserción. Se asume que los registros siempre se insertan al final del archivo. Es necesario leer la última página del archivo, añadir el registro y escribir la página. El coste es $2D + C$.

Borrado. Debe encontrarse el registro, eliminarlo de la página y escribirla en disco. Para simplificar, se supone que no se intenta compactar el archivo para liberar el espacio libre generado por los borrados¹. El coste es el de la búsqueda más $C + D$.

Se asume que el registro a borrar está especificado usando el identificador de registro. Como la página puede obtenerse fácilmente desde el identificador de registro, se puede leer directamente la página. Por tanto, el coste de la búsqueda es D .

Si el registro a borrar está especificado mediante una condición de igualdad o por rango sobre algunos campos, el coste de la búsqueda es el indicado en la descripción de selección de igualdad y rango. El coste del borrado también está afectado por el número de registros seleccionados, pues deben modificarse todas las páginas que contengan estos registros.

9.4.3 Archivos ordenados

Exploración. El coste es $B(D + RC)$ porque todas las páginas deben examinarse. Obsérvese que este caso no es mejor ni peor que en los archivos no ordenados. Sin embargo, los registros se recuperan según el criterio de ordenación, es decir, en orden por *edad*, y para una edad determinada, en orden por *sueldo*.

Búsqueda con selección de igualdad. Supóngase que la selección de igualdad se corresponde con el criterio de ordenación (*edad, sueldo*). En otras palabras, la condición de selección se especifica sobre al menos el primer campo de la clave compuesta (por ejemplo, *edad = 30*). Si no, (por ejemplo, *sueldo = 50* o *departamento="Juguetes"*), el criterio de ordenación no ayuda y el coste es idéntico al de un archivo de montículo.

Se puede localizar la primera página que contenga el registro o registros deseados, si existen, con una búsqueda binaria en $\log_2 B$ pasos. (Este análisis asume que las páginas de un archivo ordenado se almacenan secuencialmente, y que es posible recuperar la i -ésima página del archivo directamente en una operación de E/S.) Cada paso requiere una operación de E/S y dos comparaciones. Una vez que se conoce la página, el primer registro que participe en la selección puede localizarse haciendo de nuevo una búsqueda binaria de la página con un coste de $Clog_2 R$. El coste es $Dlog_2 B + Clog_2 R$, lo que supone una mejora significativa sobre la búsqueda en archivos de montículo.

Si varios registros satisfacen la condición (por ejemplo, “Determinar todos los empleados de 18 años”) está garantizado que son adyacentes debido a la ordenación por *edad*, y por tanto el coste de recuperar todos los registros es el coste de localizar el primero ($Dlog_2 B + Clog_2 R$) más el coste de leer todos los registros en orden secuencial. Normalmente, todos los registros que satisfacen la condición caben en una sola página. Si ningún registro satisface la condición, debe buscarse la página que debería contener el primer registro de la búsqueda, si existiera, y buscar en dicha página.

Búsqueda con selección por rango. De nuevo se asume que la selección por rango se corresponde con la clave compuesta, y el primer registro que satisface la selección se localiza como en el caso anterior. Posteriormente, las páginas de datos se recuperan secuencialmente

¹En la práctica se utiliza un directorio u otra estructura de datos para seguir la pista del espacio libre, de forma que los registros se inserten en el primer hueco libre. Esto incrementa un poco el coste de la inserción y el borrado, pero no lo suficiente como para afectar a la comparación.

hasta que se encuentra un registro que no satisface la selección por rango; es similar a una búsqueda por igualdad con muchos registros que satisfacen la condición.

El coste total es el coste de búsqueda más el coste de recuperación del conjunto de registros que satisfacen la búsqueda. El coste de la búsqueda incluye el coste de lectura de la primera página que contiene los registros que satisfacen la selección. Para selecciones por rangos pequeñas, todos los registros seleccionados aparecerán en esta página. Para selecciones por rangos mayores, habrá que leer páginas adicionales que contengan los registros que satisfagan la búsqueda.

Inserción. Para insertar un registro y mantener el orden, es necesario encontrar primero la posición correcta en el archivo, añadir el registro, y después leer y reescribir todas las páginas posteriores (pues todos los registros se desplazan una posición, suponiendo que no hay posiciones vacías). En promedio se puede suponer que el registro insertado se encuentra en la mitad del archivo. Por tanto, es necesario leer la segunda mitad del archivo y escribirla de nuevo después de añadir el nuevo registro. El coste es el correspondiente a encontrar la posición del registro nuevo más $2(0,5B(D + RC))$, es decir, el coste de búsqueda más $B(D + RC)$.

Borrado. Es necesario buscar el registro, borrarlo de la página y escribir la página modificada. También deben leerse y escribirse todas las páginas posteriores porque todos los registros que siguen al registro eliminado deben desplazarse para compactar el espacio libre². El coste es el mismo que para una inserción, es decir, el coste de búsqueda más $B(D + RC)$. Dado el idr del registro a borrar, es posible leer directamente la página que contiene el registro.

Si los registros a eliminar se especifican mediante una condición de igualdad o por rango, el coste de borrado depende del número de registros que satisfacen la condición. Si la condición se especifica sobre el campo de ordenación los registros serán contiguos, y el primer registro puede localizarse utilizando búsqueda binaria.

9.4.4 Archivos agrupados

Un estudio experimental extensivo ha comprobado que en un archivo agrupado las páginas tienen normalmente alrededor de un 67 por 100 de ocupación. Por ello, el número de páginas físicas de datos es aproximadamente $1,5B$, y utilizaremos esta observación en el siguiente análisis.

Exploración. El coste de una exploración es $1,5B(D + RC)$ porque deben examinarse todas las páginas de datos; esto es similar a los archivos ordenados, con el ajuste evidente del mayor número de páginas de datos. Obsérvese que nuestra métrica de coste no captura las diferencias potenciales de coste debidas a las operaciones de E/S secuencial. Cabe esperar que los archivos ordenados sean superiores en este punto, aunque un archivo agrupado que use ISAM se aproximaría a ellos (más que los árboles B+).

Búsqueda con selección de igualdad. Se supondrá que la selección de igualdad se corresponde con la clave de búsqueda $\langle \text{edad}, \text{sueldo} \rangle$. Es posible localizar la primera página que contiene el registro o registros deseados (si existen) en $\log_2 1,5B$ pasos, es decir, leyendo todas las páginas desde la raíz a la hoja correspondiente. En la práctica, la página raíz

²A diferencia del archivo de montículo, no hay forma económica de gestionar el espacio libre, por lo que se toma en consideración el coste de compactación de un archivo cuando se elimina un registro.

probablemente estará en la memoria intermedia y se ahorrará una operación de E/S, pero se ignorará esto en el análisis simplificado. Cada paso requiere una operación de E/S y dos comparaciones. Una vez que se conoce la página, el primer registro seleccionado se puede localizar utilizando búsqueda binaria de la página con un coste $Clog_2R$. El coste por tanto es $Dlog_{1,5}B + Clog_2R$, lo que representa una mejora significativa incluso respecto a la búsqueda en archivos ordenados.

Si varios registros satisfacen la condición (por ejemplo, “Determinar todos los empleados de 18 años”), serán adyacentes debido a la ordenación por *edad*, y por tanto el coste de recuperar todos los registros es el coste de localizar el primero ($Dlog_{1,5}B + Clog_2R$) más el coste de leer todos los demás registros en orden secuencial.

Búsqueda con selección por rango. Suponiendo de nuevo que la selección por rango corresponde a la clave compuesta, el primer registro que satisface la selección se localiza como en la búsqueda por igualdad. Después, las páginas de datos se recuperan secuencialmente (utilizando los enlaces siguiente y anterior en el nivel de las hojas) hasta que se encuentra un registro que no satisface la selección por rango; esto es similar a una búsqueda por igualdad con muchos registros que satisfacen la condición.

Inserción. Para insertar un registro, primero hay que encontrar la página hoja correcta en el índice, leyendo cada página desde la raíz. Después, hay que añadir el registro. La mayor parte de las veces la página hoja tiene espacio suficiente para el nuevo registro, y todo lo que hay que hacer es escribir la página hoja modificada. Ocasionalmente la página estará llena y habrá que recuperar y modificar otras páginas, pero esto es suficientemente poco frecuente como para ignorarlo en este análisis simplificado. El coste es por tanto el coste de la búsqueda más una escritura, $Dlog_{1,5}B + Clog_2R + D$.

Borrado. Debe buscarse el registro, eliminarlo de la página, y escribir la página modificada. La discusión y el análisis de coste de inserción se aplica también aquí.

9.4.5 Archivo de montículo con índice en árbol no agrupado

El número de páginas hoja en un índice depende del tamaño de una entrada de datos. Se supone que cada entrada de datos en el índice es una décima parte del tamaño de un registro de empleados, lo que es habitual. El número de páginas hoja en el índice es $0,1(1,5B) = 0,15B$, si se tiene en cuenta la ocupación del 67 por 100 de las páginas de índice. De igual modo, el número de entradas de datos en una página es $10(0,67R) = 6,7R$, teniendo en cuenta el tamaño relativo y la ocupación.

Exploración. Considérese la Figura 9.1 que representa un índice no agrupado. Para realizar una exploración completa del archivo de registros de empleados, se puede explorar el nivel de hojas del índice y, para cada entrada de datos, leer el registro de datos correspondiente del archivo, obteniendo los registros de datos ordenados por $\langle edad, sueldo \rangle$.

Es posible leer todas las entradas de datos con un coste de $0,15B(D + 6,7RC)$ operaciones de E/S. Ahora viene la parte más cara: hay que leer el registro de empleados para cada entrada de datos del índice. El coste de leer los registros de empleados es una operación de E/S por registro, pues el índice no está agrupado y cada entrada de datos de una página hoja del índice puede apuntar a una página diferente en el archivo de empleados. El coste de este paso es $BR(D + C)$, prohibitivamente alto. Si se desea obtener los registros de empleados

en orden, sería mejor ignorar el índice y explorar el archivo de empleados directamente y ordenarlo. Una regla general simple es que un archivo se puede ordenar con un algoritmo de dos pasadas en el que cada pasada requiere leer y escribir el archivo entero. Por tanto, el coste de E/S de ordenar un archivo con B páginas es $4B$, mucho menos que el coste de utilizar un índice no agrupado.

Búsqueda con selección de igualdad. Supóngase que la selección de igualdad se corresponde con el criterio de ordenación $\langle \text{edad}, \text{suelto} \rangle$. Se puede localizar la primera página que contiene la entrada o entradas de datos deseadas (si existen) en $\log_2 0,15B$ pasos, es decir, leyendo todas las páginas desde la raíz hasta la hoja apropiada. Cada paso requiere una operación de E/S y dos comparaciones. Una vez que se conoce la página, la primera entrada de datos de la selección se puede localizar mediante una búsqueda binaria dentro de la página con un coste $C\log_2 6,7R$. El primer registro de datos se puede leer del archivo de empleados con otra operación de E/S. El coste por tanto es $D\log_2 0,15B + C\log_2 6,7R + D$, una mejora significativa sobre la búsqueda en archivos ordenados.

Si hay varios registros que satisfacen la condición (por ejemplo, “Determinar todos los empleados de 18 años”), no se garantiza que sean adyacentes. El coste de recuperar todos los registros es el coste de localizar la primera entrada de datos ($D\log_2 0,15B + C\log_2 6,7R$) más una operación de E/S por cada registro seleccionado. Por tanto, el coste de utilizar un índice no agrupado depende mucho del número de registros seleccionados.

Búsqueda con selección por rango. De nuevo suponiendo que la selección por rango se corresponde con la clave compuesta, el primer registro que satisface la selección se localiza de la misma forma que con la búsqueda por igualdad. Posteriormente, las entradas de datos se recuperan secuencialmente (utilizando los enlaces siguiente y anterior en el nivel de las hojas del índice) hasta que se encuentra una entrada de datos que no satisface la selección por rango. Para cada entrada de datos seleccionada, se realiza una operación de E/S para leer los registros correspondientes de empleados. El coste puede hacerse prohibitivo rápidamente según aumenta el número de registros que satisface la selección por rango. Como norma general, si el 10 por 100 de los registros satisface la condición, es mejor recuperar todos los registros, ordenarlos y quedarse con los que cumplen la condición.

Inserción. Primero es necesario insertar el registro en el archivo de montículo de empleados, con un coste de $2D + C$. Además, debe insertarse la entrada de datos correspondiente en el índice. Encontrar la hoja correspondiente tiene un coste $D\log_2 0,15B + C\log_2 6,7R$, y escribirla después de añadir la nueva entrada de datos tiene un coste adicional D .

Borrado. Se debe localizar el registro de datos en el archivo de empleados y la entrada de datos en el índice, con un coste $D\log_2 0,15B + C\log_2 6,7R + D$. Después es necesario escribir las páginas modificadas en los archivos de índice y datos, con un coste $2D$.

9.4.6 Archivo de montículo con índice asociativo no agrupado

Siguiendo el mismo tratamiento de los índices en árbol no agrupados, supóngase que cada entrada de datos ocupa una décima parte del tamaño de un registro de datos. Se conside-

rará solamente la asociatividad estática en este análisis, y por simplicidad se supondrá que no hay cadenas de desbordamiento³.

En un archivo asociativo estático las páginas se mantienen con una ocupación de alrededor del 80 por 100 (para dejar espacio para inserciones futuras y minimizar los desbordamientos cuando el archivo crece). Esto se consigue añadiendo una nueva página a un cajón cuando cada página existente está llena al 80 por 100 al cargar inicialmente los registros en el archivo asociativo. El número de páginas requerido para almacenar las entradas de datos es por tanto 1,25 veces el número de páginas necesarias cuando está densamente compactado, es decir, $1,25(0,10B) = 0,125B$. El número de entradas de datos que caben en una página es $10(0,80R) = 8R$, teniendo en cuenta el tamaño relativo y la ocupación.

Exploración. Como en el caso anterior, todas las entradas de datos se pueden recuperar sin mucho coste: $0,125B(D + 8RC)$ operaciones de E/S. No obstante, por cada entrada se produce un coste adicional de una operación de E/S para leer el registro de datos correspondiente; el coste de este paso es $BR(D + C)$. Esto es un coste desorbitado y, más aún, los resultados no están ordenados. Por esto nadie explora un índice asociativo.

Búsqueda con selección de igualdad. Esta operación se realiza muy eficientemente para selecciones con condiciones de igualdad para cada campo de la clave compuesta $\langle\text{edad}, \text{sueldo}\rangle$. El coste de identificar la página que contiene entradas de datos seleccionadas es H . Suponiendo que este cajón está formado por una página solamente (es decir, sin páginas de desbordamiento), recuperarlo tiene un coste D . Si se supone que se encuentra la entrada de datos después de explorar la mitad de los registros de la página, el coste de explorar la página es $0,5(8R)C = 4RC$. Finalmente, hay que leer el registro de datos del archivo de empleados, lo que tiene un coste de otro D . El coste total es por tanto $H + 2D + 4RC$, que es aún más bajo que el coste de un índice en árbol.

Si varios registros satisfacen la condición, *no* se garantiza que sean adyacentes. El coste de recuperarlos todos es el coste de localizar la primera entrada de datos ($H + D + 4RC$) más una operación de E/S por cada registro seleccionado. Por tanto, el coste de usar un índice no agrupado depende mucho del número de registros seleccionados.

Búsqueda con selección por rango. La estructura asociativa no proporciona ayuda en este caso, y debe explorarse todo el archivo de montículo de empleados con un coste $B(D + RC)$.

Inserción. Primero debe insertarse el registro en el archivo de montículo, con un coste $2D + C$. Además, debe localizarse la página apropiada en el índice, modificarse para insertar la nueva entrada y escribirse. El coste adicional es $H + 2D + C$.

Borrado. Es necesario localizar el registro de datos en el archivo de empleados y la entrada de datos en el índice; este paso de búsqueda tiene un coste $H + 2D + 4RC$. Después deben escribirse las páginas modificadas en ambos archivos, con un coste $2D$.

³Las variantes dinámicas de asociatividad son menos sensibles al problema de las cadenas de desbordamiento, y tienen un coste medio por búsqueda ligeramente superior, pero por lo demás son similares a la versión estática.

9.4.7 Comparación de los costes de E/S

La Figura 9.4 compara los costes de E/S para las distintas organizaciones de archivos que se han estudiado. Un archivo de montículo tiene una buena eficiencia y permite una rápida exploración e inserción de registros. Sin embargo, es lento para las búsquedas y borrados.

<i>Tipo de archivo</i>	<i>Exploración</i>	<i>Búsq. por igualdad</i>	<i>Búsqueda por rango</i>	<i>Inserción</i>	<i>Borrado</i>
Montículo	BD	$0,5BD$	BD	$2D$	$Búsq. + D$
Ordenado	BD	$D\log_2 B$	$D\log_2 B + \text{núm. págs. sel.}$	$Búsq. + BD$	$Búsq. + BD$
Agrupado	$1,5BD$	$D\log_{F1,5} B$	$D\log_{F1,5} B + \text{núm. págs. sel.}$	$Búsq. + D$	$Búsq. + D$
Índice en árbol no agrupado	$BD(R + 0,15)$	$D(1 + \log_{F0,15} B)$	$D(\log_{F0,15} B + \text{núm. págs. sel.})$	$D(3 + \log_{F0,15} B)$	$Búsq. + 2D$
Índice asociativo no agrupado	$BD(R+0,125)$	$2D$	BD	$4D$	$Búsq. + 2D$

Figura 9.4 Comparación de costes de E/S

Un archivo ordenado también tiene una buena eficiencia de almacenamiento, pero la inserción y el borrado de registros son lentos. Las búsquedas son más rápidas que con los archivos de montículo. Es importante indicar que en un SGBD real un archivo casi nunca se mantiene completamente ordenado.

Un archivo agrupado ofrece todas las ventajas de un archivo ordenado y trata eficientemente inserciones y borrados. (Estas ventajas suponen una sobrecarga de espacio respecto al archivo ordenado, pero se compensa con creces.) Las búsquedas son aún más rápidas que en los archivos ordenados, aunque un archivo ordenado puede ser más rápido si se recupera un gran número de registros secuencialmente, debido a la mayor eficiencia de las operaciones de E/S con bloques.

Los índices en árbol y asociativo no agrupados proporcionan búsquedas, inserciones y borrados rápidos, pero las exploraciones y las búsquedas por rangos con muchos registros seleccionados son lentas. Los índices asociativos son un poco más rápidos en las búsquedas por igualdad, pero no admiten búsquedas por rangos.

En resumen, la Figura 9.4 demuestra que ninguna organización de archivo es uniformemente superior en todas las situaciones.

9.5 ÍNDICES Y AJUSTE DEL RENDIMIENTO

En este apartado se ofrece una visión general de las alternativas disponibles cuando se utilizan índices para mejorar el rendimiento de un sistema de base de datos. La elección de los índices

tiene un impacto enorme en el rendimiento del sistema, y debe realizarse en el contexto de la **carga de trabajo** (*workload*) esperada, es decir, la combinación más habitual de operaciones de consulta y actualización.

El tratamiento completo de los índices y su rendimiento requiere entender la evaluación de consultas de la base de datos y el control de concurrencia. Por ello se recuperará este tema en el Capítulo 20, que se basa en la discusión de este apartado. En particular, se introducirán ejemplos con múltiples tablas en el Capítulo 20 porque requieren comprender los algoritmos de reunión y los planes de evaluación de consultas.

9.5.1 Impacto de la carga de trabajo

Lo primero a considerar es la carga de trabajo esperada y las operaciones comunes. Como se ha visto, las distintas organizaciones de archivos e índices soportan bien diferentes operaciones.

En general, un índice soporta eficientemente la recuperación de entradas de datos que satisfacen una condición de selección dada. Recuérdese del apartado anterior que hay dos tipos importantes de selecciones: las selecciones de igualdad y las selecciones por rangos. Las técnicas de indexación asociativas están optimizadas solamente para selecciones de igualdad y son poco eficientes para las selecciones por rangos, donde normalmente son peores que explorar todo el archivo de registros. Las técnicas de indexación basadas en árboles soportan ambos tipos de condiciones de selección eficientemente, lo que explica su uso generalizado.

Tanto los índices en árbol como los asociativos pueden soportar inserciones, borrados y actualizaciones bastante eficientemente. Los índices basados en árboles en particular ofrecen una alternativa superior para mantener archivos de registros completamente ordenados. En contraste con el mantenimiento simple de entradas de datos en un archivo ordenado, la discusión de los índices estructurados en árbol (B+) del Apartado 9.3.2 destaca dos ventajas importantes sobre los archivos ordenados:

1. Se pueden manejar inserciones y borrados de datos eficientemente.
2. Es mucho más rápido encontrar la página hoja correcta cuando se busca un registro por valor de clave que mediante una búsqueda binaria en un archivo ordenado.

La desventaja relativa es que las páginas de un archivo ordenado pueden estar ubicadas físicamente en orden en el disco, haciendo mucho más rápida la recuperación de páginas en orden secuencial. Por supuesto, las inserciones y borrados sobre un archivo ordenado son extremadamente costosas. Una variante de los árboles B+, denominada método de acceso secuencial indexado (Indexed Sequential Access Method, ISAM), ofrece la ventaja de la ubicación secuencial de páginas hoja, además de búsquedas rápidas. Las inserciones y borrados no se gestionan tan bien como en los árboles B+, pero son mucho mejores que en un archivo ordenado. La indexación estructurada en árboles se estudiará en detalle en el Capítulo 10.

9.5.2 Organización de índices agrupados

Como se vio en el Apartado 9.2.1, un índice agrupado realmente es una organización de archivo para los registros de datos subyacentes. Los registros de datos pueden ser grandes, y debería evitarse su repetición; así que puede haber como mucho un índice agrupado para una

colección de registros dada. Por otra parte, se pueden construir varios índices no agrupados en un archivo de datos. Supóngase que los registros de empleados están ordenados por *edad*, o almacenados en un archivo agrupado con clave de búsqueda *edad*. Si además hay un índice por el campo *sueldo*, no debe ser agrupado. También se puede construir un índice no agrupado por, por ejemplo, *númd*, si existe este campo.

Los índices agrupados son costosos de mantener, aunque menos que un archivo completamente ordenado. Cuando se debe insertar un nuevo registro en una página hoja llena, debe asignarse una nueva página hoja y algunos de los registros existentes tienen que trasladarse a esta página. Si los registros están identificados por una combinación de identificador de página y posición, como es habitual en los sistemas de base de datos actuales, todos los sitios de la base de datos que apuntan a un registro que se mueve a la nueva página (normalmente, entradas de otros índices para la misma colección de registros) también deben actualizarse para apuntar a la nueva ubicación. Localizar todos estos sitios y hacer estas modificaciones adicionales puede suponer varias operaciones de E/S. El agrupamiento debe utilizarse pocas veces y solamente cuando esté justificado por consultas frecuentes que se beneficien de él. En particular, no hay ninguna buena razón para construir un archivo agrupado utilizando asociatividad, pues las consultas por rangos no pueden consultarse utilizando índices asociativos.

Cuando se trata la limitación de que sólo un índice puede ser agrupado, a menudo es útil considerar si la información en la clave de búsqueda de un índice es suficiente para contestar la consulta. Si es así, los sistemas de base de datos modernos son suficientemente inteligentes como para evitar leer los registros de datos. Por ejemplo, si hay un índice por *edad* y se debe calcular la media de edad de los empleados, el SGBD puede hacerlo simplemente examinando las entradas de datos en el índice. Esto es un ejemplo de una **evaluación de índice**. En una evaluación de índice de una consulta no se necesita acceder a los registros de datos en los archivos que contienen las relaciones de la consulta; es posible evaluar completamente la consulta mediante los índices de los archivos. Un beneficio importante de la evaluación de índice es que tiene la misma eficiencia con índices no agrupados, pues sólo se utilizan las entradas de datos en las consultas. De esta forma, se pueden utilizar los índices no agrupados para acelerar ciertas consultas si se sabe que el SGBD explotará la evaluación de índices.

Ejemplos de diseño de índices agrupados

Para ilustrar el uso de un índice agrupado en una consulta por rango, considérese el siguiente ejemplo:

```
SELECT    E.númd
FROM      Empleados E
WHERE     E.edad > 40
```

Si se tiene un índice de árbol B+ sobre *edad* se puede usar para recuperar exclusivamente las tuplas que satisfacen la selección *E.edad>40*. Que este índice valga la pena depende en primer lugar de la selectividad de la condición: qué porcentaje de los empleados es mayor de 40. Si prácticamente todos son mayores de 40, se gana poco con un índice por *edad*; una exploración secuencial de la relación tardaría casi lo mismo. Sin embargo, supóngase que solamente un 10 por 100 de los empleados son mayores de 40. ¿Sería útil ahora este índice?

La respuesta depende de si el índice está agrupado. Si no lo está, es posible necesitar una operación de E/S de una página por cada empleado seleccionado, y esto puede ser más costoso que una exploración secuencial, ¡incluso si se selecciona sólo el 10 por 100 de los empleados! Por otra parte, un índice de árbol B+ por *edad* sólo requiere el 10 por 100 de las operaciones de E/S de una exploración secuencial (ignorando las operaciones de E/S necesarias para recorrer el árbol desde la raíz a la primera página hoja y las operaciones para las páginas hoja relevantes del índice).

En otro ejemplo, considérese el siguiente refinamiento de la consulta anterior:

```
SELECT E.númd, COUNT(*)
FROM Empleados E
WHERE E.edad > 10
GROUP BY E.númd
```

Si se encuentra disponible un índice de árbol B+ por *edad*, se pueden recuperar con él las tuplas, ordenarlas respecto a *númd* y de esta forma contestar la consulta. Sin embargo, esto puede no ser un buen plan si casi todos los empleados tienen más de 10 años. Y este plan es especialmente malo si el índice no está agrupado.

A continuación se analizará si un índice por *númd* podría ajustarse mejor a este propósito. Se puede utilizar este índice para recuperar todas las tuplas agrupadas por *númd*, y por cada *númd* contar el número de tuplas con *edad* > 10. (Esta estrategia se puede utilizar tanto con índices asociativos como de árbol B+; sólo es necesario que las tuplas estén *agrupadas*, no necesariamente *ordenadas*, por *númd*.) La eficiencia de nuevo depende decisivamente de si el índice está agrupado. Si lo está, este plan es probablemente el mejor si la condición por *edad* no es muy selectiva. (Incluso si se tiene un índice agrupado por *edad*, si la condición no es selectiva, el coste de ordenar las tuplas seleccionadas por *númd* será probablemente alto.) Si el índice no está agrupado, se puede realizar una operación de E/S por tupla en Empleados, y esto sería terrible. De hecho, si el índice no está agrupado el optimizador elegirá el plan directo basado en ordenar por *númd*. Por tanto, esta consulta sugiere que se cree un índice agrupado por *númd* si la condición sobre *edad* no es muy selectiva. Si lo es, en su lugar se debería crear un índice (no necesariamente agrupado) por *edad*.

El agrupamiento también es importante para un índice sobre una clave de búsqueda que no incluya una clave candidata, es decir, un índice en el que varias entradas de datos puedan tener el mismo valor de clave. Para ilustrar este punto considérese la siguiente consulta:

```
SELECT E.númd
FROM Empleados E
WHERE E.hobby='Sellos'
```

Si mucha gente colecciona sellos, la recuperación de tuplas mediante un índice no indexado por *hobby* puede ser muy ineficiente. Es posible que sea menos costoso explorar la relación para recuperar todas las tuplas y aplicar la selección sobre la marcha. Por tanto, si la consulta es importante, se debería considerar hacer este índice agrupado. Por otra parte, si se asume que *ide* es una clave de Empleados y se sustituye la condición *E.hobby='Sellos'* por *E.ide=552*, sabemos que como mucho una tupla de Empleados satisfará esta condición. En este caso, no hay ninguna ventaja en hacer el índice agrupado.

La siguiente consulta muestra cómo pueden influir las operaciones de agregación en la elección de índices:

```
SELECT E.númd, COUNT(*)
FROM Empleados E
GROUP BY E.númd
```

Un plan directo para esta consulta consiste en ordenar Empleados por *númd* para calcular el número de empleados para cada *númd*. Sin embargo, si está disponible un índice —asociativo o de árbol B+— se puede contestar esta consulta explorando solamente el índice. Para cada valor de *númd*, se cuenta simplemente el número de entradas de datos en el índice con este valor en la clave de búsqueda. Obsérvese que no importa si el índice está agrupado porque no se van a recuperar tuplas de Empleados.

9.5.3 Claves de búsqueda compuestas

La clave de búsqueda de un índice puede contener varios campos; estas claves se denominan **claves de búsqueda compuestas** o **claves concatenadas**. Como ejemplo, considérese una colección de registros de empleados, con los campos *nombre*, *edad* y *sueldo*, ordenada por *nombre*. La Figura 9.5 muestra la diferencia entre un índice compuesto con clave *<edad, sueldo>*, un índice compuesto con clave *<sueldo, edad>*, un índice con clave *edad* y un índice con clave *sueldo*. Todos los índices mostrados utilizan la alternativa (2) para las entradas de datos.

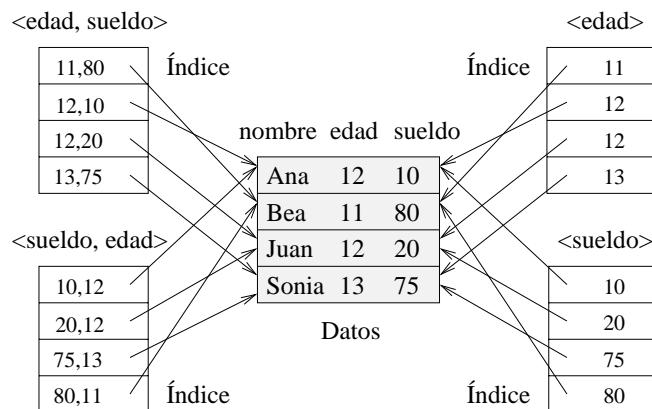


Figura 9.5 Índices de clave compuesta

Considerando una clave de búsqueda compuesta, una **consulta de igualdad** es la que *cada* campo de la clave de búsqueda está ligado a una constante. Por ejemplo, se puede pedir la recuperación de todas las entradas de datos con *edad* = 20 y *sueldo* = 10. La organización de archivo asociativa sólo permite consultas de igualdad, pues una función de asociación identifica el cajón que contiene los registros deseados sólo si se especifica un valor para cada campo de la clave de búsqueda.

Respecto a un índice de clave compuesta, en una **consulta por rango** no todos los campos de la clave de búsqueda están ligados a constantes. Por ejemplo, se puede pedir

la recuperación de todas las entradas de datos con $edad = 20$; esta consulta implica que cualquier valor es aceptable para el campo *sueldo*. Otro ejemplo de consulta por rango: se puede solicitar recuperar todas las entradas de datos con $edad < 30$ y $sueldo > 40$.

Obsérvese que el índice no puede ayudar en la consulta $sueldo > 40$ porque, intuitivamente, el índice organiza los registros primero por *edad* y después por *sueldo*. Si no se especifica *edad*, los registros a seleccionar pueden estar repartidos por todo el índice. Se dice que un índice se **corresponde** con una condición de selección si se puede utilizar para recuperar exactamente las tuplas que satisfacen la condición. Para selecciones de la forma $condición \wedge \dots \wedge condición$ se puede definir cuándo un índice se corresponde con la selección de la siguiente forma: para un índice asociativo, una selección se corresponde con el índice si incluye una condición de igualdad (“campo = constante”) por cada campo de la clave de búsqueda compuesta del índice. Para un índice en árbol se corresponde si incluye una condición de igualdad o por rango sobre un *prefijo* de la clave de búsqueda compuesta. (Como ejemplos, $\langle edad \rangle$ y $\langle edad, sueldo, númd \rangle$ son prefijos de la clave $\langle edad, sueldo, númd \rangle$, pero $\langle edad, númd \rangle$ y $\langle sueldo, númd \rangle$ no lo son.)

Compromisos en la elección de claves compuestas

Un índice de clave compuesta puede soportar un rango mayor de consultas porque se corresponde con más condiciones de selección. Más aún, como las entradas de datos de un índice compuesto contienen más información sobre el registro de datos (es decir, más campos que un índice de un atributo), las posibilidades de aplicar estrategias de evaluación de índice se incrementan. (Recuérdese del Apartado 9.5.2 que una evaluación de índice no necesita acceder a los registros de datos, pues encuentra todos los valores de campos en las entradas de datos de los índices.)

En el lado negativo, los índices compuestos deben actualizarse con cualquier operación (inserción, borrado o actualización) que modifique *cualquier* campo de la clave de búsqueda. Los índices compuestos son también más grandes que un índice de clave con un solo atributo porque el tamaño de las entradas es mayor. En el caso de un índice compuesto de árbol B+ esto también implica un posible incremento del número de niveles, aunque se puede usar compresión de claves para mitigar este problema (véase el Apartado 10.8.1).

Ejemplos de diseño de claves compuestas

Considérese la siguiente consulta, que devuelve todos los empleados con $20 < edad < 30$ y $3000 < sueldo < 5000$:

```
SELECT E.ide
  FROM Empleados E
 WHERE E.edad BETWEEN 20 AND 30
   AND E.sueldo BETWEEN 3000 AND 5000
```

Un índice compuesto por $\langle edad, sueldo \rangle$ podría servir si las condiciones de la cláusula WHERE fueran algo selectivas. Obviamente, un índice asociativo no ayudará; se requiere un índice de árbol B+ (o ISAM). También está claro que un índice agrupado será probablemente superior a un índice no agrupado. Para esta consulta, en la que las condiciones sobre *edad* y *sueldo* son

igualmente selectivas, un índice de árbol B+ agrupado y compuesto por $\langle\text{edad}, \text{suelto}\rangle$ es tan efectivo como el mismo tipo de índice, pero sobre $\langle\text{suelto}, \text{edad}\rangle$. No obstante, el orden de los atributos de búsqueda a veces puede ser muy relevante, como se muestra en la siguiente consulta:

```
SELECT E.ide
FROM Empleados E
WHERE E.edad = 25
AND E.suelto BETWEEN 3000 AND 5000
```

En esta consulta un índice compuesto de árbol B+ y agrupado por $\langle\text{edad}, \text{suelto}\rangle$ tendrá un buen rendimiento porque los registros están ordenados primero por *edad* y después (si dos registros tienen el mismo valor para *edad*) por *suelto*. Así, todos los registros con *edad* = 25 están agrupados. Por otra parte, un índice del mismo tipo por $\langle\text{suelto}, \text{edad}\rangle$ no es tan eficiente. En este caso, los registros se ordenan primero por *suelto*, y por tanto dos registros con el mismo valor para *edad* (en particular, con *edad* = 25) pueden estar muy separados. En efecto, este índice permite utilizar la selección por rango por *suelto*, pero no la selección de igualdad por *edad*. (Se puede obtener un buen rendimiento de ambas variantes de la consulta utilizando un único índice *espacial*.)

Los índices compuestos también son útiles cuando se tratan muchas consultas de agregación. Considérese:

```
SELECT AVG (E.suelto)
FROM Empleados E
WHERE E.edad = 25
AND E.suelto BETWEEN 3000 AND 5000
```

Un índice de árbol B+ compuesto sobre $\langle\text{edad}, \text{suelto}\rangle$ permite contestar la consulta con una exploración del índice. El mismo tipo de índice sobre $\langle\text{suelto}, \text{edad}\rangle$ también lo permite, pero se recuperan más entradas de índice en este caso que con el primero.

El siguiente ejemplo es una variación de un ejemplo anterior:

```
SELECT E.númd, COUNT(*)
FROM Empleados E
WHERE E.suelto=10.000
GROUP BY E.númd
```

Un índice por *númd* solamente no permite evaluar esta consulta con una exploración de índice, porque es necesario ver el valor del campo *suelto* de cada tupla para verificar que *suelto* = 10.000. Sin embargo, se puede utilizar un plan sólo de índice si se tiene un índice de árbol B+ compuesto sobre $\langle\text{suelto}, \text{númd}\rangle$ o $\langle\text{númd}, \text{suelto}\rangle$. En un índice con clave $\langle\text{suelto}, \text{númd}\rangle$ todas las entradas de datos con *suelto* = 10.000 son contiguas (tanto si el índice está agrupado como si no). Más aún, estas entradas están ordenadas por *númd*, lo que permite obtener fácilmente la cuenta de cada grupo de *númd*. Obsérvese que solamente necesitamos recuperar las entradas de datos con *suelto* = 10.000.

Es importante señalar que esta estrategia no funciona si la cláusula WHERE se modifica para utilizar la condición *suelto* > 10.000. Aunque es suficiente para recuperar entradas de datos de índice —es decir, se aplicaría una estrategia de sólo índice— estas entradas deben

ordenarse ahora por $númd$ para identificar los grupos (porque, por ejemplo, dos entradas con el mismo valor $númd$ pero diferente valor para *sueldo* pueden no ser contiguas). Un índice con clave $\langle númd, sueldo \rangle$ es mejor para esta consulta: las entradas de datos con un valor determinado para $númd$ están ordenadas juntas, y cada uno de estos grupos de entradas está ordenado por *sueldo*. Para cada grupo de $númd$, se pueden eliminar las entradas con *sueldo* no mayor a 10.000 y contar el resto. (Usar este índice es menos eficiente que una exploración de índice con clave $\langle sueldo, númd \rangle$ para la consulta con *sueldo* = 10.000, porque deben leerse todas las entradas de datos. Así, la elección entre estos índices está influenciada por la consulta que sea más común.)

Supóngase en el siguiente ejemplo que se desea encontrar el mínimo *sueldo* de cada $númd$:

```
SELECT E.númd, MIN(E.sueldo)
FROM Empleados E
GROUP BY E.númd
```

Un índice por $númd$ solamente no permite evaluar esta consulta con una exploración de índice. Sin embargo, se puede utilizar un plan de índice si se tiene un índice de árbol B+ compuesto sobre $\langle númd, sueldo \rangle$. Obsérvese que todas las entradas de datos del índice con un valor de $númd$ determinado están almacenadas juntas (tanto si el índice es agrupado como si no). Más aún, este grupo de entradas está ordenado por *sueldo*. Un índice por $\langle sueldo, númd \rangle$ permite evitar recuperar los registros de datos, pero las entradas de datos del índice deben estar ordenadas por $númd$.

9.5.4 Especificación de índices en SQL:1999

Una cuestión natural en este punto es cómo se pueden crear índices utilizando SQL. El estándar SQL:1999 *no* incluye ninguna sentencia para crear o eliminar estructuras de índices. De hecho, ¡el estándar ni siquiera requiere que las implementaciones de SQL soporten índices! En la práctica, por supuesto, todos los SGBD relacionales comerciales soportan uno o más tipos de índices. El siguiente comando para crear un índice de árbol B+ —los índices de árbol B+ se describen en el Capítulo 10— es ilustrativo:

```
CREATE INDEX IndClasifEdad ON Alumnos
    WITH STRUCTURE = BTREE,
        KEY = (edad, nota)
```

Esta sentencia especifica que se va a crear un índice de árbol B+ sobre la tabla Alumnos usando como clave la concatenación de las columnas *edad* y *nota*. De este modo, los valores de clave son pares de la forma $\langle edad, nota \rangle$, y hay una entrada distinta para cada uno de estos pares. Una vez que se ha creado, el SGBD mantiene el índice añadiendo o eliminando entradas de datos en respuesta a las inserciones o borrados de registros en la relación Alumnos.

9.6 PREGUNTAS DE REPASO

Las respuestas a las preguntas de repaso se pueden encontrar en los apartados indicados.

- ¿Dónde almacena un SGBD datos persistentes? ¿Cómo lleva los datos a memoria principal para su procesamiento? ¿Qué componente del SGBD lee y escribe los datos de memoria principal, y qué es la unidad de E/S? (**Apartado 9.1**)
- ¿Qué es una *organización de archivo*? ¿Qué es un *índice*? ¿Cuál es la relación entre archivos e índices? ¿Se pueden tener varios índices sobre un único archivo de registros? ¿Puede un índice almacenar registros de datos (es decir, actuar como un archivo)? (**Apartado 9.2**)
- ¿Qué es la *clave de búsqueda* de un índice? ¿Qué es una *entrada de datos* de un índice? (**Apartado 9.2**)
- ¿Qué es un índice *agrupado*? ¿Qué es un *índice primario*? ¿Cuántos índices agrupados se pueden crear sobre un archivo? ¿Cuántos índices no agrupados se pueden crear? (**Apartado 9.2.1**)
- ¿Cómo se organizan los datos en un índice asociativo? ¿Cuándo se usaría un índice asociativo? (**Apartado 9.3.1**)
- ¿Cómo se organizan los datos en un índice en árbol? ¿Cuándo se usaría un índice en árbol? (**Apartado 9.3.2**)
- Considérense las siguientes operaciones: *exploraciones, selecciones de igualdad y rango, inserciones y borrados*, y las siguientes organizaciones de archivo: *archivos de montículo, ordenados, agrupados, archivos de montículo con índice no agrupado en árbol sobre la clave de búsqueda y archivos de montículo con índice asociativo no agrupado*. ¿Qué organización se adapta mejor a cada operación? (**Apartado 9.4**)
- ¿Cuáles son los factores que contribuyen más al coste de las operaciones de base de datos? Explíquese un modelo de coste sencillo que refleje esto. (**Apartado 9.4.1**)
- ¿Cómo influye la carga de trabajo prevista en las decisiones de diseño físico de la base de datos, tales como los índices a crear? ¿Por qué la elección de los índices es un aspecto fundamental del diseño físico de bases de datos? (**Apartado 9.5**)
- ¿Qué problemas se consideran cuando se utilizan índices agrupados? ¿Qué es un método de *evaluación de índice*? ¿Cuál es su ventaja principal? (**Apartado 9.5.2**)
- ¿Qué es una *clave de búsqueda compuesta*? ¿Cuáles son las ventajas y los inconvenientes de las claves de búsqueda compuestas? (**Apartado 9.5.3**)
- ¿Qué comandos SQL permiten la creación de índices? (**Apartado 9.5.4**)

EJERCICIOS

Ejercicio 9.1 Contéstese a las siguientes preguntas sobre datos de un SGBD en almacenamiento externo:

1. ¿Por qué un SGBD guarda los datos en un almacenamiento externo?
2. ¿Por qué son importantes los costes de E/S en un SGBD?

<i>ide</i>	<i>nombre</i>	<i>usuario</i>	<i>edad</i>	<i>nota</i>
53831	Martín	martin@musica	11	1,8
53832	Gómez	gomez@musica	12	2,0
53666	Juanes	juanes@inf	18	3,4
53688	Ortiz	ortiz@ie	19	3,2
53650	Ortiz	ortiz@mat	19	3,8

Figura 9.6 Un ejemplar de la relación Alumnos, ordenada por *edad*

3. ¿Qué es un identificador de registro? Dado un identificador de registro, ¿cuántas operaciones de E/S se necesitan para llevarlo a memoria principal?
4. ¿Cuál es la función del gestor de la memoria intermedia de un SGBD? ¿Cuál es la función del gestor de espacio en disco? ¿Cómo interactúan estas capas con la capa de archivos y métodos de acceso?

Ejercicio 9.2 Contéstese a las siguientes preguntas sobre archivos e índices:

1. ¿Qué operaciones soporta la abstracción de archivo de registros?
2. ¿Qué es un índice sobre un archivo de registros? ¿Qué es una clave de búsqueda de un índice? ¿Por qué son necesarios los índices?
3. ¿Qué alternativas están disponibles para las entradas de datos en un índice?
4. ¿Cuál es la diferencia entre un índice primario y uno secundario? ¿Qué es una entrada de datos duplicada en un índice? ¿Puede una clave primaria contener duplicados?
5. ¿Cuál es la diferencia entre un índice agrupado y uno no agrupado? Si un índice contiene registros de datos como “entradas de datos”, ¿puede ser no agrupado?
6. ¿Cuántos índices agrupados se pueden crear sobre un archivo? ¿Se debería crear al menos un índice agrupado para cada archivo?
7. Considérense las alternativas (1), (2) y (3) para las ‘entradas de datos’ de un índice, como se vio en el Apartado 9.2. ¿Son todas ellas apropiadas para los índices secundarios? Razónese la respuesta.

Ejercicio 9.3 Considérese una relación almacenada en un archivo ordenado aleatoriamente para el que el único índice es un índice no agrupado sobre un campo denominado *sueldo*. Si desea recuperar todos los registros con *sueldo* > 20, ¿utilizar el índice es siempre la mejor alternativa? Razónese la respuesta.

Ejercicio 9.4 Considérese el ejemplar de la relación Alumnos mostrado en la Figura 9.6, ordenado por *edad*. Para el objetivo de esta pregunta, supóngase que estas tuplas están almacenadas en un archivo ordenado en ese orden: la primera tupla está en la página 1, la segunda también está en la página 1, etcétera. Cada página puede almacenar tres registros de datos, por lo que la cuarta tupla está en la página 2.

Explíquese el contenido de las entradas de datos en cada uno de los siguientes índices. Si el orden de las entradas es significativo, indíquese y explíquese el porqué. Si el índice no puede construirse, indíquese y explíquese el porqué.

1. Un índice no agrupado sobre *edad* utilizando la alternativa (1).
2. Un índice no agrupado sobre *edad* utilizando la alternativa (2).
3. Un índice no agrupado sobre *edad* utilizando la alternativa (3).
4. Un índice agrupado sobre *edad* utilizando la alternativa (1).
5. Un índice agrupado sobre *edad* utilizando la alternativa (2).
6. Un índice agrupado sobre *edad* utilizando la alternativa (3).
7. Un índice no agrupado sobre *nota* utilizando la alternativa (1).

8. Un índice no agrupado sobre *nota* utilizando la alternativa (2).
9. Un índice no agrupado sobre *nota* utilizando la alternativa (3).
10. Un índice agrupado sobre *nota* utilizando la alternativa (1).
11. Un índice agrupado sobre *nota* utilizando la alternativa (2).
12. Un índice agrupado sobre *nota* utilizando la alternativa (3).

Ejercicio 9.5 Explíquese la diferencia entre los índices asociativos y los índices en árbol B+. En particular, coméntese cómo funcionan las búsquedas de igualdad y de rango utilizando un ejemplo.

Ejercicio 9.6 Rellénense los costes de E/S de la Figura 9.7.

Tipo de archivo	Exploración	Búsqueda de igualdad	Búsqueda por rango	Inserción	Borrado
Montículo					
Ordenado					
Agrupado					
Índice de árbol no agrupado					
Índice asociativo no agrupado					

Figura 9.7 Comparación de costes de E/S

Ejercicio 9.7 Si se fuera a crear un índice para una relación, ¿qué consideraciones guiarían la elección? Coméntese:

1. La elección de índice primario.
2. Índices agrupados frente a no agrupados.
3. Índices asociativos frente a índices de árbol.
4. El uso de un archivo ordenado mejor que un índice basado en árbol.
5. Elección de la clave de búsqueda del índice. ¿Qué es una clave de búsqueda compuesta y qué consideraciones se tienen en cuenta al elegir claves compuestas? ¿Qué son los planes sólo de índice y cuál es la influencia de planes de evaluación de índice en la elección de la clave de búsqueda?

Ejercicio 9.8 Considérese una operación de borrado especificada mediante una condición de igualdad. Para cada una de las cinco organizaciones de archivo, ¿cuál es el coste si no hay ningún registro seleccionado? ¿cuál es el coste si la condición no está en una clave?

Ejercicio 9.9 ¿Cuáles son las conclusiones principales que se pueden obtener de las cinco organizaciones de archivo básicas descritas en el Apartado 9.4 ? ¿Cuáles se deberían elegir para un archivo en el que las operaciones más frecuentes son las siguientes?

1. Búsqueda de registros basada en un rango de valores.
2. Realización de inserciones y exploraciones, en las que el orden de los registros no es relevante.
3. Búsqueda de un registro basada en el valor de un campo particular.

Ejercicio 9.10 Considérese la siguiente relación:

`Emp(ide: integer, sueldo: integer, edad: real, idd: integer)`

Hay un índice agrupado sobre *ide* y un índice no agrupado sobre *edad*.

1. ¿Cómo se utilizarían los índices para hacer cumplir la restricción de que *ide* sea clave?

2. Dese un ejemplo de una actualización que sea *definitivamente acelerada* por los índices disponibles (una descripción de la actualización es suficiente).
3. Dese un ejemplo de una actualización que vaya *definitivamente más despacio* por los índices (una descripción de la actualización es suficiente).
4. ¿Se puede encontrar un ejemplo de una actualización que no se vea afectada por los índices?

Ejercicio 9.11 Considérense las siguientes relaciones:

`Emp(ide: integer, nombree: varchar, sueldo: integer, edad: integer, idd: integer)`
`Dept(idd: integer, presupuesto: integer, planta: integer, ide_director: integer)`

Los sueldos van de 10.000 € a 100.000 €, las edades están entre 20 y 80 años, cada departamento tiene en promedio cinco empleados, hay 10 plantas, y los presupuestos están entre 10.000 € y un millón de euros. Se puede suponer una distribución uniforme de los valores.

Para cada una de las siguientes consultas, ¿qué elección de índices se debería elegir para acelerarla? Si el sistema de base de datos no considera planes de sólo índice (es decir, se accede siempre a los registros de datos incluso si hay suficiente información en la entrada del índice), ¿cómo cambiaría la respuesta? Razónese brevemente.

1. Consulta: *Imprimir nombree, edad y sueldo de todos los empleados.*
 - (a) Índice asociativo agrupado sobre los campos $\langle \text{nombree}, \text{edad}, \text{sueldo} \rangle$ de Emp.
 - (b) Índice asociativo no agrupado sobre los campos $\langle \text{nombree}, \text{edad}, \text{sueldo} \rangle$ de Emp.
 - (c) Índice de árbol B+ agrupado sobre los campos $\langle \text{nombree}, \text{edad}, \text{sueldo} \rangle$ de Emp.
 - (d) Índice asociativo no agrupado sobre los campos $\langle \text{ide}, \text{idd} \rangle$ de Emp.
 - (e) Sin índice.
2. Consulta: *Determinar los idd de los departamentos que estén en la décima planta y tengan un presupuesto de menos de 15.000 €.*
 - (a) Índice asociativo agrupado sobre el campo *planta* de Dept.
 - (b) Índice asociativo no agrupado sobre el campo *planta* de Dept.
 - (c) Índice de árbol B+ agrupado sobre los campos $\langle \text{planta}, \text{presupuesto} \rangle$ de Dept.
 - (d) Índice de árbol B+ agrupado sobre el campo *presupuesto* de Dept.
 - (e) Sin índice.



EJERCICIOS BASADOS EN PROYECTOS

Ejercicio 9.12 Contéstese a las siguientes preguntas:

1. ¿Qué técnicas de indexación están soportadas en Minibase?
2. ¿Qué alternativas están soportadas para entradas de datos?
3. ¿Se permiten índices agrupados?

NOTAS BIBLIOGRÁFICAS

Varios libros estudian la organización de archivos en detalle [18, 205, 281, 331, 386, 419, 472].

Las notas bibliográficas de índices asociativos y árboles B+ se incluyen en los Capítulos 10 y 11.



10

ÍNDICES DE ÁRBOL

- ☞ ¿Cuál es la intuición que está detrás de los índices estructurados en árbol? ¿Por qué son buenos para las selecciones por rangos?
- ☞ ¿Cómo maneja un índice ISAM las búsquedas, inserciones y borrados?
- ☞ ¿Cómo maneja un índice de árbol B+ las búsquedas, inserciones y borrados?
- ☞ ¿Cuál es el impacto de los valores de clave duplicados en la implementación de índices?
- ☞ ¿Qué es la compresión de claves y por qué es importante?
- ☞ ¿Qué es la carga masiva (bulk-loading), y por qué es importante?
- ☞ ¿Qué le ocurre a los identificadores de registros cuando se actualizan índices dinámicos? ¿Cómo afecta esto a los índices agrupados?
- ➡ **Conceptos clave:** ISAM, índices estáticos, páginas de desbordamiento, problemas de bloqueo; árboles B+, índices dinámicos, equilibrio (balance), conjuntos de secuencias, formato de nodo; operación de inserción en árboles B+, división de nodos, operación de borrado, combinación frente a redistribución, ocupación mínima; duplicados, páginas de desbordamiento, inclusión de idr en claves de búsqueda; compresión de claves; carga masiva; efectos de la división por idr en índices agrupados.

El que quiera la fruta debe trepar al árbol.

— Thomas Fuller

En este capítulo se consideran dos estructuras de datos de índices, denominadas ISAM y árboles B+, basadas en organizaciones en árbol. Estas estructuras proporcionan soporte eficiente de búsquedas por rangos, incluyendo como caso especial las exploraciones de archivos

ordenados. A diferencia de los archivos ordenados, estas estructuras de índice soportan inserción y borrado eficientes. También permiten realizar selecciones de igualdad, aunque no son tan eficientes en este caso como los índices asociativos, que se tratan en el Capítulo 11.

Un árbol ISAM¹ es una estructura de índice estática que resulta efectiva cuando el archivo no se actualiza frecuentemente, pero que no es adecuada para archivos que crecen y disminuyen mucho. Se verá ISAM en el Apartado 10.2. El árbol B+ es una estructura dinámica que se ajusta con elegancia a los cambios en el archivo. Es la estructura de índice que se utiliza más porque se ajusta bien a los cambios y permite consultas tanto de igualdad como por rangos. Se introducen los árboles B+ en el Apartado 10.3. Los árboles B+ se verán en detalle en los apartados siguientes. El Apartado 10.3.1 describe el formato de un nodo de un árbol. En el Apartado 10.4 se considera la búsqueda de registros utilizando un índice de árbol B+. El Apartado 10.5 presenta el algoritmo para insertar registros en un árbol B+, y en el 10.6 se presenta el algoritmo de borrado. El Apartado 10.7 describe la gestión de duplicados. El capítulo concluye con una discusión sobre algunos problemas prácticos relacionados con los árboles B+ en el Apartado 10.8.

Notación: En las estructuras ISAM y B+, las páginas hoja contienen *entradas de datos*, de acuerdo a la terminología utilizada en el Capítulo 9. Por convenio, se denotará con $k*$ una entrada de datos con valor de clave de búsqueda k . Las páginas que no son hojas contienen *entradas de índice* de la forma \langle valor de clave de búsqueda, identificador de página \rangle , y se utilizan para dirigir la búsqueda de una determinada entrada de datos (que está almacenada en alguna página hoja). A menudo se usará simplemente *entrada* cuando el contexto deja clara la naturaleza de la entrada (de índice o de datos).

10.1 INTRODUCCIÓN A LOS ÍNDICES DE ÁRBOL

Considérese un archivo de registros de alumnos ordenado por *nota*. Para contestar a una selección por rango como: “Determinar todos los alumnos con un nota mayor que 3,0”, se debe identificar el primero de estos alumnos haciendo una búsqueda binaria del archivo y después explorar el archivo desde ese punto. Si el archivo es grande, la búsqueda binaria inicial puede ser bastante costosa, pues el coste es proporcional al número de páginas leídas. ¿Sería posible mejorar este método?

Una idea podría consistir en crear un segundo archivo con un registro por cada página del archivo (de datos) original, de la forma \langle primera clave de la página, puntero a la página \rangle , también ordenado por el atributo clave (que en el ejemplo es *nota*). El formato de una página en el segundo archivo *índice* está ilustrado en la Figura 10.1.

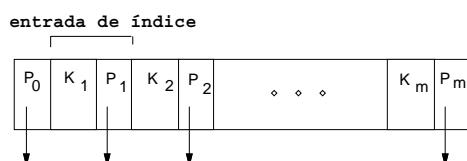


Figura 10.1 Formato de una página de índice

¹ISAM son las siglas de método de acceso secuencial indexado (Indexed Sequential Access Method).

Nos referiremos a pares de la forma $\langle \text{clave}, \text{puntero} \rangle$ como *entradas de índice* o simplemente *entradas* cuando el contexto está claro. Obsérvese que cada página de índice contiene un puntero más que el número de claves —cada clave sirve de *separador* de los contenidos de las páginas apuntadas por los punteros a su izquierda y derecha—.

Esta sencilla estructura de datos de archivo de índice se muestra en la Figura 10.2.

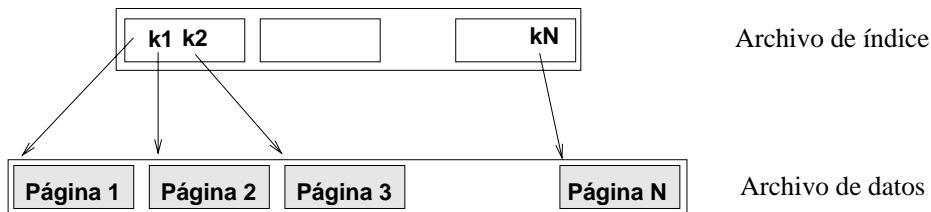


Figura 10.2 Estructura de índice de un nivel

Es posible realizar una búsqueda binaria del archivo de índice para identificar la página que contiene el primer valor de clave (*nota*) que satisface la selección por rango (en el ejemplo, el primer alumno con *nota* por encima de 3,0) y seguir el puntero a la página que contiene el registro correspondiente. Después se puede explorar el archivo de datos secuencialmente desde ese punto para recuperar los demás registros seleccionados. Este ejemplo utiliza el índice para encontrar la primera página de datos que contiene un registro de alumnos con *nota* mayor a 3,0, y el archivo de datos se explora desde ese punto para recuperar el resto de registros de alumnos.

Como el tamaño de una entrada en el archivo de índice (valor de clave e identificador de página) es probablemente mucho más pequeño que el tamaño de la página, y sólo existe una de estas entradas por cada página del archivo de datos, el archivo de índice será probablemente mucho más pequeño que el archivo de datos. Por tanto, una búsqueda binaria del archivo de índice es mucho más rápida que sobre el archivo de datos. Sin embargo, una búsqueda binaria del archivo de índice podría ser aún bastante costosa, y el archivo de índice podría ser todavía suficientemente grande como para hacer las inserciones y los borrados costosos.

El tamaño potencialmente grande del archivo de índice motiva la idea de indexación en árbol: ¿por qué no aplicar el paso anterior para construir una estructura auxiliar sobre la colección de registros de *índice* y así recursivamente hasta que la estructura auxiliar más pequeña quepa en una página? Esta construcción repetida de un índice de un nivel lleva a una estructura en árbol con varios niveles de páginas que no son hojas.

Como se indicó en el Apartado 9.3.2, la potencia de este enfoque viene del hecho de que ubicar un registro (dado un valor de clave de búsqueda) implica un recorrido de la raíz a una hoja, con una operación de E/S por nivel (como mucho; algunas páginas, como por ejemplo la raíz, están probablemente en la memoria intermedia). Dado el valor típico del grado de salida (por encima de 100), los árboles raramente tienen más de 3-4 niveles.

El siguiente aspecto a considerar es la forma en que la estructura del árbol puede manejar inserciones y borrados de entradas de datos. Se han utilizado dos enfoques distintos, dando lugar a las estructuras de datos ISAM y árboles B+, que se comentarán en las siguientes secciones.

10.2 MÉTODO DE ACCESO SECUENCIAL INDEXADO (ISAM)

La estructura de datos ISAM se muestra en la Figura 10.3. Las entradas de datos de un índice ISAM están en las páginas hoja del árbol y en páginas de *desbordamiento* adicionales encadenadas a alguna página hoja. Los sistemas de base de datos organizan cuidadosamente la disposición de las páginas de forma que los límites de las páginas se correspondan estrechamente con las características físicas del dispositivo de almacenamiento en el que residan. La estructura ISAM es completamente estática (excepto por las páginas de desbordamiento, de las que se espera que sean pocas) y facilita este tipo de optimizaciones de bajo nivel.

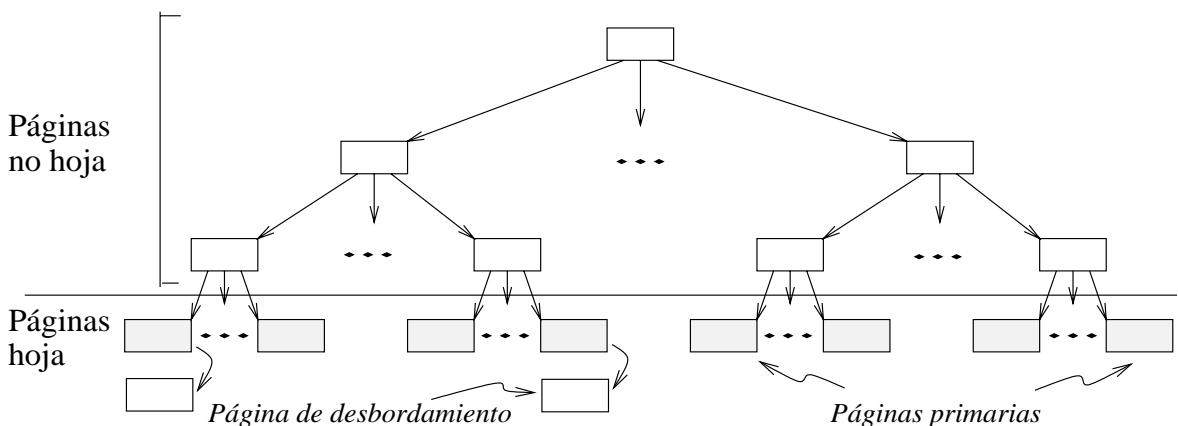


Figura 10.3 Estructura de índice ISAM

Cada nodo del árbol es una página de disco, y todos los datos residen en páginas hoja. Esto corresponde a un índice que utiliza la alternativa (1) para las entradas de datos, en términos de las alternativas descritas en el Capítulo 9; es posible crear un índice con la alternativa (2) almacenando los registros de datos en un archivo separado y almacenando pares $\langle \text{clave}, \text{idr} \rangle$ en las páginas hoja del índice ISAM. Cuando se crea el archivo, todas las páginas hoja se sitúan secuencialmente y ordenadas por valor de clave de búsqueda. (Si se usan las alternativas (2) o (3), los registros de datos se crean y ordenan antes de crear las páginas hoja del índice ISAM.) Después se crean las páginas intermedias. Si hay varias inserciones posteriores en el archivo, de forma que se inserten más entradas en una hoja de las que caben en una sola página, se necesitan páginas adicionales porque la estructura de índice es estática. Estas páginas se crean en un área de desbordamiento. La localización de páginas se muestra en la Figura 10.4.

Las operaciones básicas de inserción, borrado y búsqueda son todas completamente directas. Para una búsqueda de selección de igualdad, se comienza en el nodo raíz y se determina el subárbol en el que buscar comparando el valor del campo de búsqueda del registro con los valores de clave en el nodo. (El algoritmo de búsqueda es idéntico al de un árbol B+, que se presenta en detalle más adelante.) Para una consulta por rango, el punto inicial en el nivel de datos (u hoja) se determina de forma similar, y a continuación se recuperan las páginas de datos secuencialmente. Para las inserciones y los borrados, se determina la página

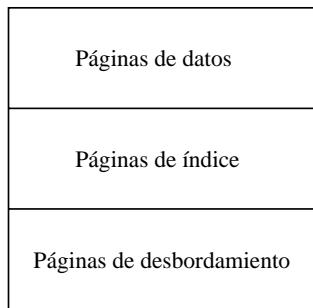


Figura 10.4 Ubicación de páginas en ISAM

apropiada como en la búsqueda, y el registro se inserta o se elimina, añadiendo páginas de desbordamiento si es necesario.

El siguiente ejemplo ilustra la estructura de un índice ISAM. Considérese el árbol mostrado en la Figura 10.5. Todas las búsquedas comienzan en la raíz. Por ejemplo, para localizar un registro con valor de clave 27, se comienza en la raíz y se sigue el puntero izquierdo, pues $27 < 40$. Después se sigue el puntero de en medio, pues $10 \leq 27 < 33$. Para una búsqueda por rango, primero se encuentra la primera entrada de datos que satisface la consulta como en una selección de igualdad, y después se recuperan las páginas hoja primarias secuencialmente (recuperando también las páginas de desbordamiento según se necesiten, siguiendo los punteros de las páginas primarias). Se supone que las páginas hoja primarias están situadas secuencialmente—esta suposición es razonable porque el número de estas páginas es conocido cuando se crea el árbol y no cambia posteriormente debido a inserciones y borrados— y por ello no se necesitan punteros “siguiente página hoja”.

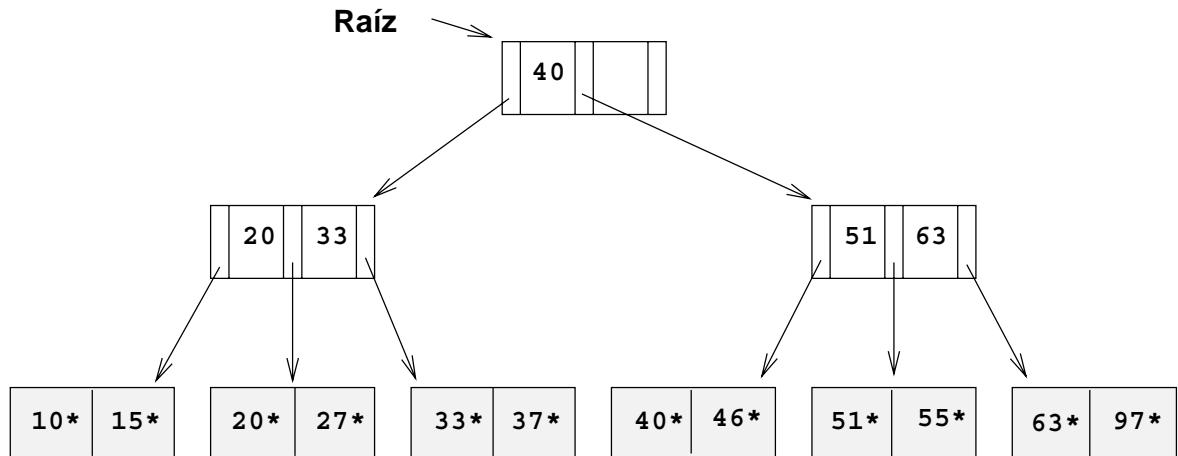


Figura 10.5 Árbol ISAM de ejemplo

Supóngase que cada página hoja puede contener dos entradas. Si se inserta un registro con valor de clave 23, la entrada 23* pertenece a la segunda página de datos, que ya contiene

20* y 27*, y no dispone de más espacio. Esta situación se trata añadiendo una página de *desbordamiento* y poniendo 23* en dicha página. Las cadenas de páginas de desbordamiento se pueden desarrollar fácilmente. Por ejemplo, insertando 48*, 41* y 42* se genera una cadena de desbordamiento de dos páginas. El árbol de la Figura 10.5 con todas estas inserciones se muestra en la Figura 10.6.

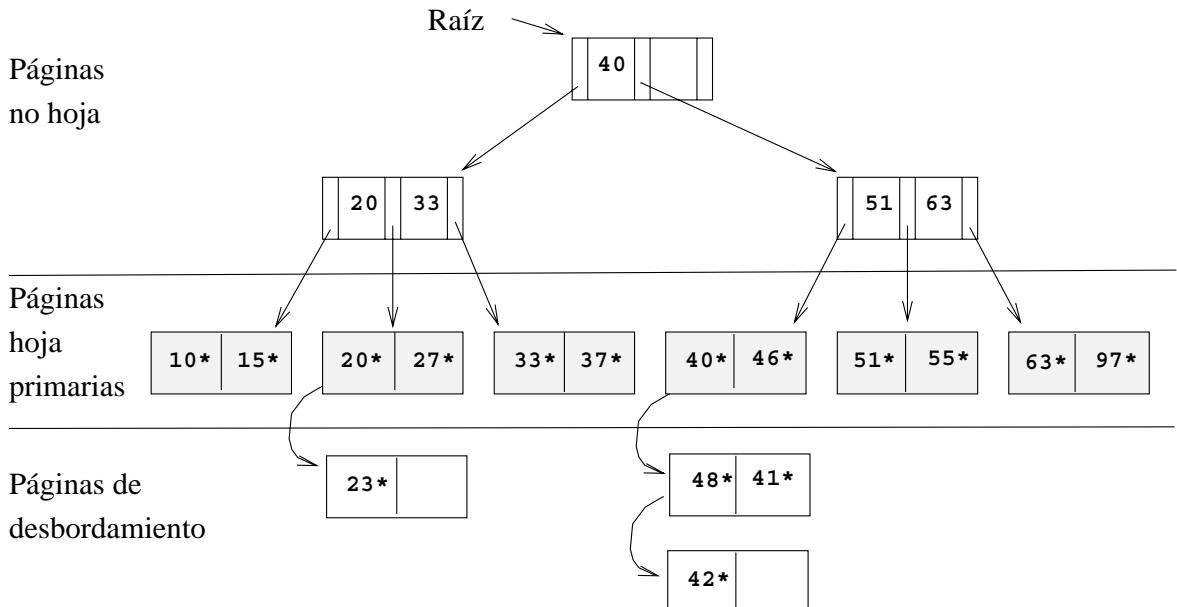


Figura 10.6 Árbol ISAM después de las inserciones

El borrado de una entrada $k*$ se realiza sencillamente eliminando dicha entrada. Si esta entrada está en una página de desbordamiento y ésta se queda vacía, la página puede ser eliminada. Si la entrada está en una página primaria y el borrado la deja vacía, la aproximación más simple es sencillamente dejar la página vacía como está; servirá como un hueco para inserciones futuras (y posiblemente páginas de desbordamiento no vacías, porque no se trasladan registros de páginas de desbordamiento a la página primaria cuando se crea espacio en ésta debido a borrados). De este modo, el número de páginas hoja primarias se fija en el momento de la creación del archivo.

10.2.1 Páginas de desbordamiento, consideraciones de bloqueo

Obsérvese que, una vez que se crea un archivo ISAM, las inserciones y los borrados solamente afectan a los contenidos de las páginas hoja. Una consecuencia de este diseño es que se podrían desarrollar largas cadenas de desbordamiento si se hace un gran número de inserciones en la misma hoja. Estas cadenas pueden afectar de forma significativa al tiempo de recuperación de un registro, pues la cadena de desbordamiento también debe recorrerse cuando una búsqueda llega a esta hoja. (Aunque los datos de una cadena de desbordamiento

se pueden mantener ordenados, normalmente no es así, para hacer las inserciones rápidas.) Para aliviar este problema, el árbol se crea inicialmente dejando alrededor del 20 por 100 de cada página libre. No obstante, una vez que se ha llenado el espacio libre con registros insertados, a no ser que se libere de nuevo con borrados, las cadenas de desbordamiento sólo pueden eliminarse mediante una reorganización completa del archivo.

El hecho de que sólo se modifiquen las páginas hoja también tiene una ventaja importante respecto a los accesos concurrentes. Cuando se accede a una página, ésta normalmente se “bloquea” por el solicitante para asegurarse de que no se va a modificar concurrentemente por otros usuarios de la página. Para modificar una página, debe bloquearse en modo “exclusivo”, lo que sólo se permite cuando nadie más bloquea la página. El bloqueo puede dar lugar a colas de usuarios (*transacciones*, para ser más precisos) esperando a tener acceso a la página. Las colas pueden ser un cuello de botella importante, especialmente para las páginas que se acceden frecuentemente, próximas a la raíz de una estructura de índice. En la estructura ISAM, como se sabe que las páginas de nivel de índice nunca se van a modificar, es posible omitir con seguridad el bloqueo. Esto es una ventaja importante de ISAM respecto a una estructura dinámica como los árboles B+. Si la distribución y el tamaño de los datos son relativamente estáticos, lo que significa que las cadenas de desbordamiento son raras, ISAM podría ser preferible a los árboles B+ debido a esta ventaja.

10.3 ÁRBOLES B+: UNA ESTRUCTURA DE ÍNDICE DINÁMICA

Una estructura estática como la de los índices ISAM tiene el problema de que se pueden crear largas cadenas de desbordamiento cuando el archivo crece, dando lugar a un rendimiento pobre. Este problema motivó el desarrollo de estructuras más flexibles y dinámicas que se ajustan de forma elegante a las inserciones y los borrados. La estructura de búsqueda en **árbol B+**, que se utiliza extensamente, es un árbol equilibrado en el que los nodos internos dirigen la búsqueda y los nodos hoja contienen las entradas de datos. Como la estructura en árbol crece y se contrae dinámicamente, no es factible asignar las páginas hoja secuencialmente como en ISAM, donde el conjunto de páginas hoja primarias era estático. Para recuperar todas las páginas hoja eficientemente, hay que enlazarlas mediante punteros de página. Organizándolas en una lista doblemente enlazada, se puede recorrer fácilmente la secuencia de páginas hoja (a menudo llamado **conjunto secuencia**) en cualquier sentido. Esta estructura se ilustra en la Figura 10.7².

Algunas de las principales características de un árbol B+ son las siguientes:

- Las operaciones (inserción, borrado) sobre el árbol lo mantienen equilibrado.
- Se garantiza una ocupación mínima de un 50 por 100 para cada nodo excepto la raíz si se implementa el algoritmo de borrado descrito en el Apartado 10.6. Sin embargo, frecuentemente el borrado se implementa simplemente localizando la entrada de datos y

²Si el árbol se ha creado por *carga masiva* (véase el Apartado 10.8.2) de un conjunto de datos existente, el conjunto secuencia puede hacerse físicamente secuencial, pero esta ordenación física se va destruyendo gradualmente según se van añadiendo y borrando nuevos datos con el tiempo.

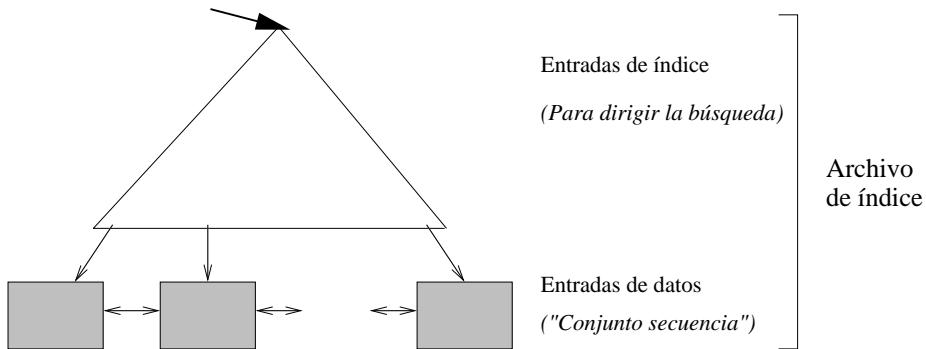


Figura 10.7 Estructura de un árbol B+

eliminándola, sin ajustar el árbol para garantizar la ocupación del 50 por 100, pues los archivos normalmente tienden a crecer más que a reducirse.

- La búsqueda de un registro requiere solamente un recorrido desde la raíz a la hoja correspondiente. La longitud de un camino desde la raíz a una hoja —cualquier hoja, porque el árbol está equilibrado— se conoce como la **altura** del árbol. Por ejemplo, un árbol con un nivel de hoja y un único nivel de índice, como el mostrado en la Figura 10.9, tiene altura 1, y un árbol que solamente tiene el nodo raíz tiene altura 0. Como tiene un grado de salida alto, la altura de un árbol B+ raramente es mayor que 3 o 4.

Se estudiarán los árboles B+ en los que cada nodo contiene m entradas, donde $d \leq m \leq 2d$. El valor d es un parámetro del árbol B+, denominado el **orden** del árbol, y es una medida de la capacidad de un nodo del árbol. El nodo raíz es la única excepción a este requerimiento del número de entradas; para la raíz, sólo se requiere que $1 \leq m \leq 2d$.

Si un archivo de registros se actualiza frecuentemente y es importante el acceso ordenado, casi siempre es mejor mantener un índice de árbol B+ con registros de datos almacenados como entradas de datos, frente a un archivo ordenado. A cambio de la sobrecarga de espacio de almacenamiento de las entradas de índice, se obtienen todas las ventajas de un archivo ordenado más unos algoritmos de inserción y borrado eficientes. Los árboles B+ típicamente mantienen un 67 por 100 de ocupación de espacio. Normalmente también son preferibles a la indexación ISAM porque las inserciones se gestionan elegantemente sin cadenas de desbordamiento. No obstante, si el tamaño del conjunto de datos y la distribución se mantienen bastante estáticas, las cadenas de desbordamiento no son un gran problema. En este caso, dos factores favorecen ISAM: las páginas hoja se asignan en secuencia (haciendo más eficientes las exploraciones sobre rangos grandes que un árbol B+, en el que las páginas probablemente no estarán en secuencia, incluso si lo estuvieron tras una carga masiva) y el coste del bloqueo con ISAM es menor que el de los árboles B+. Como regla general, sin embargo, los árboles B+ serán probablemente más eficientes que ISAM.

10.3.1 Formato de un nodo

El formato de un nodo, mostrado en la Figura 10.1, es el mismo que en ISAM. Los nodos que no son hojas con m entradas de índice contienen $m + 1$ punteros a hijos. El puntero P_i apunta a un subárbol en el que todos los valores de clave K son tales que $K_i \leq K < K_{i+1}$. Como casos especiales, P_0 apunta a un árbol en el que todos los valores de clave son menores que K_1 , y P_m apunta a un árbol en el que todos los valores de clave son mayores o iguales a K_m . En los nodos hoja las entradas se denominan $k*$, como es habitual. Exactamente igual a ISAM, los nodos hoja (y sólo los nodos hoja) contienen entradas de datos. En el caso habitual de que se utilice la alternativa (2) o (3), las entradas hoja son pares $\langle K, I(K) \rangle$, como en las entradas de nodos que no son hoja. Independientemente de la alternativa elegida para las entradas hoja, las páginas hoja están encadenadas en una lista doblemente enlazada. De este modo, las hojas forman una secuencia que puede utilizarse para contestar consultas por rangos eficientemente.

El lector debería considerar detenidamente la forma de alcanzar esta organización de los nodos con diferentes formatos de registros; a fin de cuentas, cada par clave —puntero puede entenderse como un registro—. Si el campo sobre el que se indexa es de longitud fija, estas entradas de índice serán de longitud fija; en caso contrario, serán registros de longitud variable. En cualquier caso, el árbol B+ puede verse como un archivo de registros. Si las páginas hoja no contienen los registros de datos reales, entonces el árbol B+ es realmente un archivo de registros que es distinto del archivo que contiene los datos. Si las páginas hoja contienen registros de datos, entonces el archivo contiene tanto el árbol B+ como los datos.

10.4 BÚSQUEDA

El algoritmo de búsqueda encuentra el nodo hoja al que pertenece una entrada de datos determinada. La Figura 10.8 contiene un bosquejo de este algoritmo. Se utiliza la notación $*ptr$ para indicar el valor apuntado por una variable puntero ptr y $\&(valor)$ para referirse a la dirección de *valor*. Obsérvese que encontrar i en *búsqueda_en_árbol* requiere buscar dentro del nodo, lo que se puede hacer con una búsqueda lineal o binaria (por ejemplo, dependiendo del número de entradas del nodo).

Cuando se tratan los algoritmos de búsqueda, inserción y borrado en árboles B+, se supone que no existen *duplicados*. Es decir, no se permiten dos entradas de datos con el mismo valor de clave. Por supuesto, se producen duplicados siempre que la clave de búsqueda no contiene una clave candidata, y deben tratarse en la práctica. Se verá la forma de tratar los duplicados en el Apartado 10.7.

Considérese el árbol B+ de ejemplo mostrado en la Figura 10.9. Este árbol B+ es de orden $d=2$. Es decir, cada nodo contiene entre 2 y 4 entradas. Cada entrada que no es hoja es un par $\langle valor\ de\ clave, puntero\ a\ nodo \rangle$; en el nivel de hoja, las entradas son registros de datos que se denotarán con $k*$. Para buscar la entrada 5^* , se sigue el puntero al hijo situado más a la izquierda, pues $5 < 13$. Para buscar las entradas 14^* o 15^* , debe seguirse el segundo puntero, pues $13 \leq 14 < 17$ y $13 \leq 15 < 17$. (15^* no se encuentra en la hoja hija correspondiente, y por ello puede concluirse que no está presente en el árbol.) Para encontrar 24^* , se sigue el cuarto puntero, pues $24 \leq 24 < 30$.

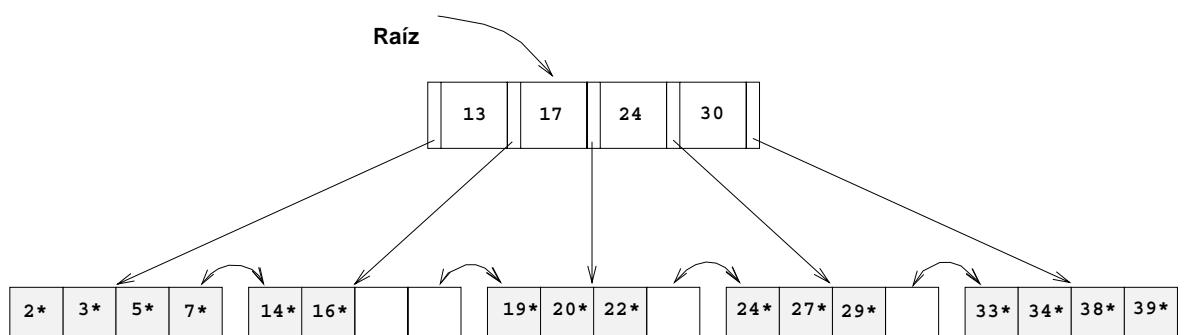
```

func encontrar (valor de clave de búsqueda  $K$ ) returns puntero_nodo
    // Dado un valor de clave de búsqueda, encuentra su nodo hoja
    devolver búsqueda_en_árbol(raíz,  $K$ );                                // busca desde la raíz
endfunc

func búsqueda_en_árbol (puntero_nodo, valor de clave de búsqueda  $K$ )
    returns puntero_nodo
    // Busca una entrada en el árbol
    si *puntero_nodo es una hoja, devolver puntero_nodo;
    si no,
        si  $K < K_1$  entonces devolver búsqueda_en_árbol( $P_0$ ,  $K$ );
        si no,
            si  $K \geq K_m$  entonces devolver búsqueda_en_árbol( $P_m$ ,  $K$ );
                //  $m =$  número de entradas
            si no,
                encontrar  $i$  tal que  $K_i \leq K < K_{i+1}$ ;
                devolver búsqueda_en_árbol( $P_i$ ,  $K$ )
endfunc

```

Figura 10.8 Algoritmo de búsqueda

Figura 10.9 Ejemplo de un árbol B+ de orden $d=2$

10.5 INSERCIÓN

El algoritmo de inserción toma una entrada, encuentra el nodo hoja al que pertenece y la inserta en dicho nodo. El pseudocódigo del algoritmo de inserción se muestra en la Figura 10.10. La idea básica que está detrás de este algoritmo es que se inserta la entrada recursivamente llamando al algoritmo de inserción en el nodo hijo apropiado. Normalmente, este procedimiento desciende al nodo hoja al que pertenece el nodo, introduce la entrada en él y vuelve atrás hasta el nodo raíz. Ocasionalmente se encuentra un nodo lleno y debe dividirse. Cuando se divide un nodo, se debe insertar en el nodo una entrada que apunte al nodo creado por la división; la variable puntero *nueva_ent.hijo* apunta a esta entrada. Si la raíz (anterior) se divide, se crea una nueva raíz y la altura del árbol se incrementa en 1.

Para ilustrar la inserción se continuará con el árbol de ejemplo mostrado en la Figura 10.9. Si se inserta la entrada 8^* , ésta pertenece a la hoja más a la izquierda, que ya está llena. Esta inserción produce una división de la página hoja; las páginas divididas se muestran en la Figura 10.11. El árbol ahora debe ajustarse para tener en cuenta la nueva página hoja, insertando una entrada con el par $\langle 5, \text{puntero a la nueva página} \rangle$ en el nodo padre. Obsérvese la copia de la clave 5, que discrimina entre la página dividida y su nueva hermana. No basta con “trasladarla”, porque cada entrada de datos debe aparecer en una página hoja.

Como el nodo padre también está lleno, se produce otra división. En general debe dividirse un nodo no hoja cuando está lleno, con $2d$ claves y $2d + 1$ punteros, y debe añadirse otra entrada de índice para tener en cuenta la división de la página hija. Ahora hay $2d + 1$ claves y $2d + 2$ punteros, lo que produce dos nodos mínimamente llenos, cada uno con d claves y $d + 1$ punteros, y una clave extra, que es la clave de “en medio”. Esta clave y un puntero al segundo nodo que no es hoja constituyen una entrada de índice que debe insertarse en el nodo padre del nodo dividido. La clave de en medio “asciende” en el árbol, a diferencia de la división de una página hoja.

Las páginas divididas de este ejemplo se muestran en la Figura 10.12. La entrada del índice que apunta al nuevo nodo que no es hoja es el par $\langle 17, \text{puntero a la nueva página de nivel de índice} \rangle$; Obsérvese que el valor de clave 17 ha “ascendido” en el árbol, a diferencia de la valor de clave de división 5 en la hoja, que fue “copiado”.

La diferencia en la gestión de divisiones de nivel de hoja y de índice viene dada por el requerimiento de los árboles B+ de que todas las entradas de datos k^* deben residir en las hojas. Este requerimiento impide que 5 “ascienda” y da lugar a la redundancia de tener algunos valores de clave tanto en el nivel de hoja como en algún nivel de índice. No obstante, las consultas por rangos pueden responderse eficientemente recuperando solamente la secuencia de páginas hoja; la redundancia es un pequeño precio a pagar por la eficiencia. Al tratar con los niveles de índice se dispone de más flexibilidad, y 17 “asciende” para evitar tener dos copias de 17 en los niveles de índice.

Ahora bien, como el nodo dividido es el antiguo nodo raíz, es necesario crear un nuevo nodo raíz que contenga la entrada que distingue entre las dos páginas de índice divididas. El árbol resultado de completar la inserción de la entrada 8^* se muestra en la Figura 10.13.

Una variante del algoritmo de inserción intenta redistribuir las entradas de un nodo N con un hermano antes de dividir el nodo; esto mejora la ocupación promedio de los nodos. El

```

proc insertar (puntero_nodo, entrada, nueva_ent_hijo)
  // Inserta una entrada en el subárbol con raíz “*puntero_nodo”;
  //el grado es d;
  //“nueva_ent_hijo” es nulo inicialmente, y es nulo cuando termina
  // a no ser que el hijo se divida

  si *puntero_nodo es un nodo no hoja, N,
    encontrar i tal que  $K_i \leq$  valor de clave de la entrada  $< K_{i+1}$ ;
    // elección de subárbol
    insertar( $P_i$ , entrada, nueva_ent_hijo);           //inserta entr. recursivamente
    si nueva_ent_hijo es nulo, terminar;             // caso normal; no divide el hijo
    si no,                                         // se divide el hijo, debe insertarse *nueva_ent_hijo en N
      si N tiene espacio disponible,                // caso normal
        introducir *nueva_ent_hijo en él,
        asignar nulo a nueva_ent_hijo, terminar;
      si no,                                     // ¡obsérvese la diferencia resp. a dividir una página hoja!
        dividir N:                                // 2d + 1 valores de clave y 2d + 2 punteros
        los primeros d valores de clave
          y d + 1 punteros a nodos permanecen,
        las últimas d claves
          y d + 1 punteros pasan al nuevo nodo, N2;
        // *nueva_ent_hijo se asigna para guiar las búsquedas
        // entre N y N2
        nueva_ent_hijo = & ((menor valor de clave en N2,
        puntero a N2));
      si N es la raíz,                            // el nodo raíz se ha dividido
        crear un nuevo nodo con ⟨puntero a N, *nueva_ent_hijo⟩;
        hacer que el puntero al nodo raíz apunte al nuevo nodo;
        terminar;

  si *puntero_nodo es un nodo hoja, L,
    si L tiene espacio disponible,              // caso normal
    introducir la entrada en él, asignar nueva_ent_hijo a nulo y terminar;
    si no,                                     // de vez en cuando el nodo hoja está lleno
      dividir L: las primeras d de entradas permanecen,
      el resto se pasan al nuevo nodo L2;
      nueva_ent_hijo = &((menor valor de clave en L2, puntero a L2));
      asignar los punteros a hermanos en L y L2;
      terminar;

endproc

```

Figura 10.10 Algoritmo de inserción en un árbol B+ de orden d

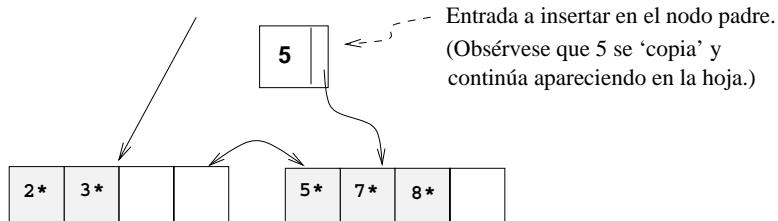


Figura 10.11 División de páginas hoja durante la inserción de la entrada 8^*

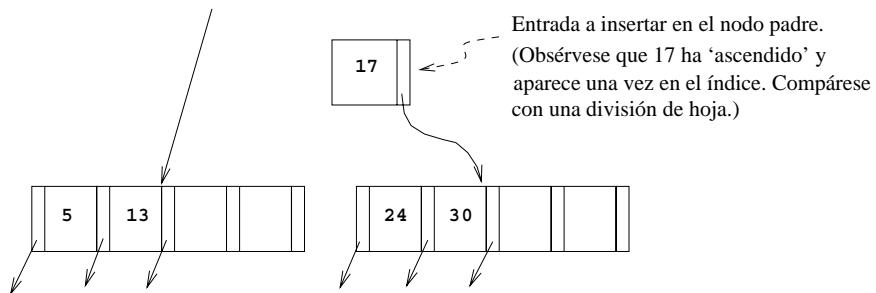


Figura 10.12 División de páginas de índice durante la inserción de la entrada 8^*

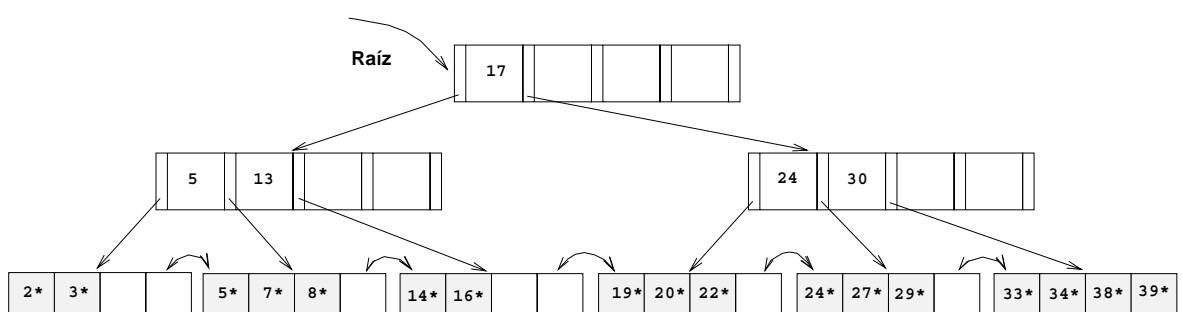


Figura 10.13 Árbol $B+$ después de insertar la entrada 8^*

hermano de un nodo N, en este contexto, es un nodo que está inmediatamente a la derecha o izquierda de N y tiene el mismo padre que N.

Para ilustrar esta redistribución, reconsidérese la inserción de la entrada 8* en el árbol mostrado en la Figura 10.9. La entrada pertenece a la hoja más a la izquierda, que está llena. Sin embargo, el (único) hermano de este nodo hoja contiene solamente dos entradas y puede por tanto acomodar más. De este modo es posible gestionar la inserción de 8* mediante una redistribución. Obsérvese el nuevo valor de clave de la entrada en el nodo padre que apunta a la segunda hoja; se “copia” el nuevo valor inferior de clave de la segunda hoja. Este proceso se muestra en la Figura 10.14.

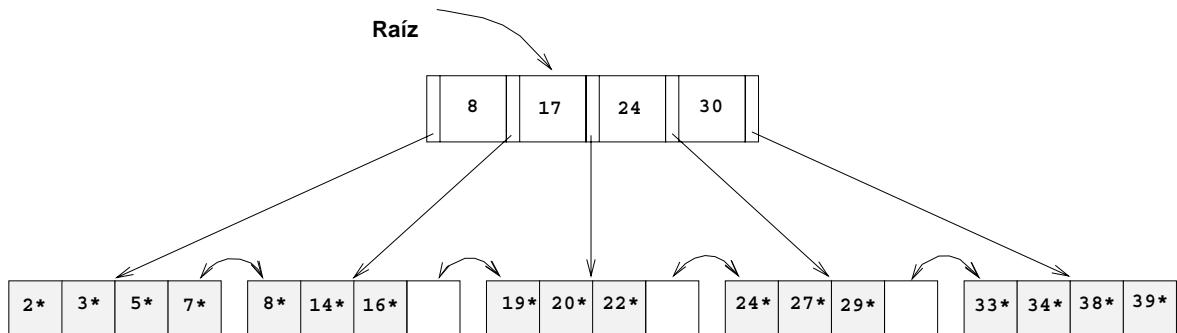


Figura 10.14 Árbol B+ después de insertar la entrada 8* utilizando redistribución

Para determinar si es posible realizar la redistribución, es necesario recuperar el nodo hermano. Si éste también está lleno, deberá dividirse el nodo de todas formas. En promedio, comprobar si es posible la redistribución incrementa el número de operaciones de E/S de división de nodos índice, especialmente si se comprueban ambos hermanos. (Comprobarlo puede reducir las operaciones de E/S si la redistribución tiene éxito cuando una división se propaga ascendentemente por el árbol, pero este caso es muy infrecuente.) Si el archivo está creciendo, la ocupación media probablemente no se verá muy afectada incluso si no se redistribuye. Tomando estas consideraciones en cuenta, normalmente compensa *no* redistribuir las entradas de los nodos que no son hojas.

Sin embargo, si se produce una división en el nivel de nodos hoja, es necesario recuperar un vecino para ajustar los punteros de vecino anterior y siguiente con respecto al nodo hoja que se acaba de crear. Por tanto, tiene sentido la redistribución de forma limitada: si el nodo hoja está lleno, se lee un nodo vecino; si éste tiene espacio y el mismo parent, se redistribuyen las entradas. En caso contrario (el vecino tiene un parent diferente, es decir, no es un hermano, o bien está también lleno) se divide el nodo hoja y se ajustan los punteros anterior y siguiente del nodo dividido, el nodo vecino recién creado y el antiguo vecino.

10.6 BORRADO

El algoritmo de borrado toma una entrada, encuentra el nodo hoja al que pertenece y la elimina. El pseudocódigo de este algoritmo se muestra en la Figura 10.15. La idea fundamental de este algoritmo es que se elimina la entrada recursivamente llamando al algoritmo de

borrado para el nodo hijo apropiado. En la mayor parte de los casos desciende al nodo hoja al que pertenece la entrada, la elimina y recorre el camino inverso al nodo raíz. Ocasionalmente un nodo está en el nivel mínimo de ocupación antes del borrado, y éste provoca que el nodo esté por debajo del umbral de ocupación. Cuando esto ocurre, es necesario redistribuir las entradas de un hermano adyacente, o bien combinar el nodo con un hermano para mantener mínima la ocupación. Si se redistribuyen las entradas entre dos nodos, el nodo padre debe actualizarse para reflejarlo; el valor de clave en la entrada de índice que apunta al segundo nodo debe cambiarse para ser el mínimo valor de clave en dicho nodo. Si se combinan dos nodos, su padre debe actualizarse eliminando la entrada de índice del segundo nodo; esta entrada de índice se referencia mediante la variable puntero *anterior_ent_hijo* cuando la llamada al algoritmo retorna al nodo padre. Si se borra la última entrada en el nodo raíz porque uno de sus hijos ha sido eliminado, la altura del árbol decrece en 1.

Para ilustrar el borrado, considérese el árbol de ejemplo mostrado en la Figura 10.13. Para eliminar la entrada 19*, basta con borrarla de la página hija en la que aparece, y no es necesario hacer más pues la hoja todavía contiene dos entradas. Sin embargo, si posteriormente se elimina 20*, la hoja pasa a contener solamente una entrada después del borrado. El (único) hermano de este nodo hoja tiene tres entradas, y por consiguiente es posible resolver la situación mediante redistribución: se traslada la entrada 24* a la página hoja que contiene 20* y se copia la nueva clave de división (27, que es el nuevo valor inferior de clave de la hoja de la que se extrajo 24*) en el padre. Este proceso se muestra en la Figura 10.16.

Supóngase que se elimina la entrada 24*. La hoja afectada contiene solamente una entrada (22*) después del borrado, y el (único) hermano contiene solamente dos entradas (27* y 29*). Por tanto, no es posible redistribuir entradas. No obstante, entre estos dos nodos hoja juntos contienen solamente tres entradas y pueden combinarse. Durante la combinación, se puede “lanzar” la entrada (*<27, puntero a la segunda página hoja*) al padre, que apuntaba a la segunda página hoja, pues está vacía después de la combinación y puede desecharse. El subárbol derecho de la Figura 10.16 después de este paso de borrado de la entrada 24* se muestra en la Figura 10.17.

El borrado de la entrada *<27, puntero a la segunda página hoja* ha creado una página intermedia con sólo una entrada, lo que está por debajo del mínimo $d = 2$. Para resolver este problema, hay que redistribuir o combinar. En cualquier caso, es necesario seleccionar un hermano. El único hermano de este nodo contiene solamente dos entradas (con valores de clave 5 y 13), por lo que no es posible redistribuir; por tanto es necesario combinar.

La situación cuando deben combinarse dos nodos intermedios es exactamente la opuesta a la situación en la que hay que dividir un nodo que no es hoja. Un nodo intermedio debe dividirse cuando contiene $2d$ claves y $2d + 1$ punteros, y es necesario añadir otro par clave-puntero. Como debe recurrirse a la combinación cuando no es posible redistribuir las entradas, los dos nodos deben tener una ocupación mínima; es decir, cada uno debe contener d claves y $d + 1$ punteros antes del borrado. Después de combinar los dos nodos y eliminar el par clave-puntero, quedarán $2d - 1$ claves y $2d + 1$ punteros: intuitivamente, el puntero más a la izquierda del segundo nodo no tiene valor de clave. Para ver el valor de clave que debe ser combinado con este puntero para crear una entrada de índice completa, considérese el padre de los dos nodos. La entrada de índice que apunta a uno de los nodos combinados debe eliminarse del padre porque el nodo va a desecharse. El valor de clave en esta entrada es precisamente el valor que es necesario completar en el nuevo nodo combinado: las entradas

```

proc borrar (puntero_padre, puntero_nodo, entrada, anterior_ent_hijo)
// Elimina una entrada del subárbol con raíz “*puntero_nodo”; el grado es d;
// “anterior_ent_hijo” es nulo inicialmente, y al finalizar a no ser que se borre el hijo
si *puntero_nodo no es un nodo hoja, N,
    encontrar i tal que  $K_i \leq$  valor de clave de la entrada  $< K_{i+1}$ ; // elige subárbol
    borrar(puntero_nodo,  $P_i$ , entrada, anterior_ent_hijo); // borrado recursivo
    si anterior_ent_hijo es nulo, terminar; // caso habitual: el hijo no se elimina
    si no, // se desecha el nodo hijo (véase el texto)
        eliminar *anterior_ent_hijo de N,
            // a continuación se comprueba si está por debajo del mínimo
        si N tiene entradas de sobra, // caso habitual
            asignar anterior_ent_hijo a nulo y terminar; // el borrado no va más allá
        si no, // ¡Obsérvese la diferencia respecto a combinar págs. hoja!
            obtener un hermano S de N:// puntero_padre se utiliza para encontrar S
            si S tiene entradas extra,
                redistribuir uniformemente entre N y S a través del padre;
                asignar anterior_ent_hijo a nulo y terminar;
            si no, combinar N y S // M es el nodo a la derecha
                anterior_ent_hijo = &(entrada en el padre para M);
                trasladar la clave de división del padre a su hijo a la izquierda;
                trasladar todas las entradas de M al nodo a su izquierda;
                desechar el nodo M vacío, terminar;

        si *puntero_nodo es un nodo hoja, L,
            si L tiene entradas de sobra, // caso habitual
                eliminar la entrada, asignar anterior_ent_hijo a nulo y terminar;
            si no, // de vez en cuando, la hoja se vacía
                obtener un hermano S de L; // puntero_padre se utiliza para encontrar S
                si S tiene entradas extra,
                    redistribuir uniformemente entre L y S;
                    encontrar la entrada en el padre para el nodo a la derecha; // llamado M
                    reemplazar el valor de clave en la entrada del padre por
                        el nuevo valor inferior en M;
                    asignar anterior_ent_hijo a nulo, terminar;
                si no, combinar L y S // M es el nodo a la derecha
                    anterior_ent_hijo = & (entrada actual en el padre para M);
                    trasladar todas las entradas de M al nodo a la izquierda;
                    desechar el nodo vacío M, ajustar los punteros a hermanos, terminar;
endproc

```

Figura 10.15 Algoritmo de borrado de un árbol B+ de orden d

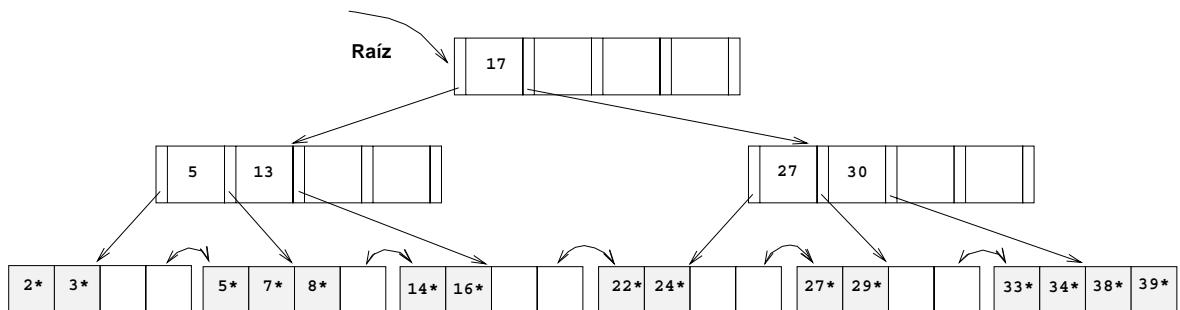


Figura 10.16 Árbol B+ después de borrar las entradas 19* y 20*

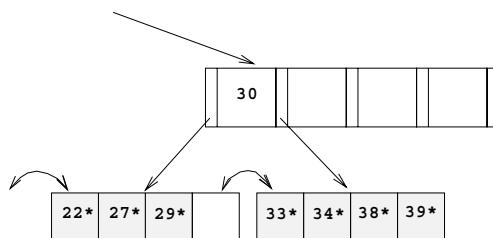


Figura 10.17 Árbol B+ parcial durante el borrado de la entrada 24*

del primer nodo, seguidas del valor de clave de división que “desciende” del padre, seguido a su vez por las entradas del segundo nodo dan un total de $2d$ claves y $2d + 1$ punteros, lo que forma un nodo intermedio lleno. Obsérvese cómo el valor de clave de división del padre “desciende”, a diferencia del caso de combinar dos nodos hoja.

Considérese la combinación de dos nodos que no son hojas en el ejemplo. Juntos, el nodo intermedio y el hermano con el que se va a combinar contienen sólo tres entradas, y tienen un total de cinco punteros a nodos hoja. Para combinar estos dos nodos, también es necesario hacer descender la entrada de índice del padre que ahora discrimina entre estos dos nodos. Esta entrada de índice tiene valor de clave 17, y de esta forma se crea una nueva entrada (*17, puntero al hijo más a la izquierda en el hermano*). Ahora se dispone de un total de cuatro entradas y cinco punteros a hijos, que pueden caber en una página de un árbol de orden $d = 2$. Obsérvese que descender la clave de división 17 quiere decir que no volverá a aparecer en el nodo padre después de la combinación. En ese momento el nodo combinado es el único hijo de la raíz anterior, que por consiguiente puede desecharse. El árbol resultante de todos estos pasos para el borrado de la entrada 24* se muestra en la Figura 10.18.

Los ejemplos anteriores han ilustrado la redistribución de entradas entre hojas y la combinación de páginas hoja e intermedias. El caso que falta es la redistribución de entradas entre páginas que no son hojas. Para comprender este caso, considérese el subárbol derecho intermedio mostrado en la Figura 10.17. Se obtendría el mismo subárbol derecho si se intentara eliminar 24* de un árbol similar al mostrado en la Figura 10.16, pero con el subárbol izquierdo y el valor de clave raíz mostrado en la Figura 10.19. Este árbol ilustra una fase intermedia durante el borrado de 24*. (El lector debería intentar construir el árbol inicial.)

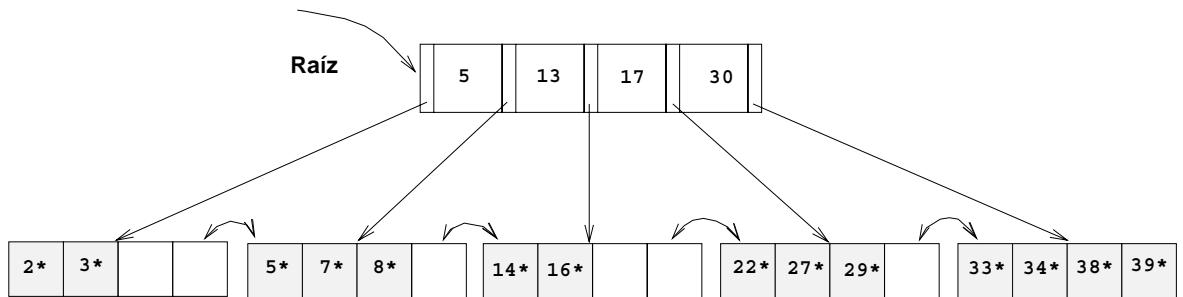


Figura 10.18 Árbol B+ después de borrar la entrada 24*

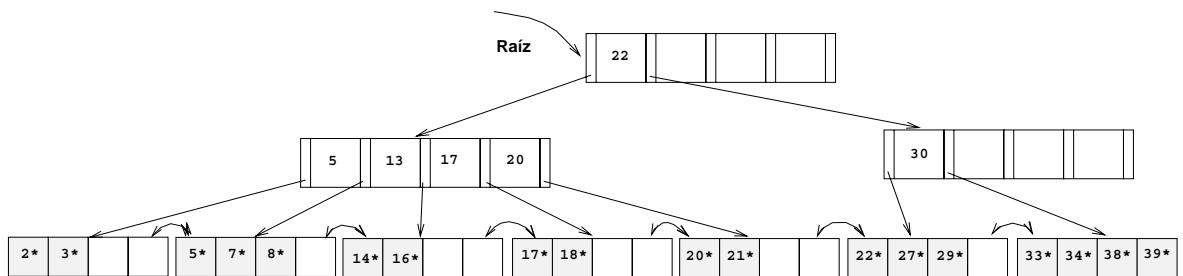


Figura 10.19 Un árbol B+ durante un borrado

A diferencia del caso en el que se eliminó 24* del árbol de la Figura 10.16, el nodo intermedio que contiene el valor de clave 30 ahora tiene un hermano que puede prestar entradas (las entradas con valores de clave 17 y 20). Se pueden trasladar esas entradas³ del hermano. Obsérvese que, al hacerlo, básicamente se pasan a través de la entrada de división del nodo padre (la raíz), que se ocupa del hecho de que 17 se convierte en el nuevo valor inferior de clave a la derecha y por tanto debe sustituir al valor de división anterior en la raíz (el valor de clave 22). El árbol con todos estos cambios se muestra en la Figura 10.20.

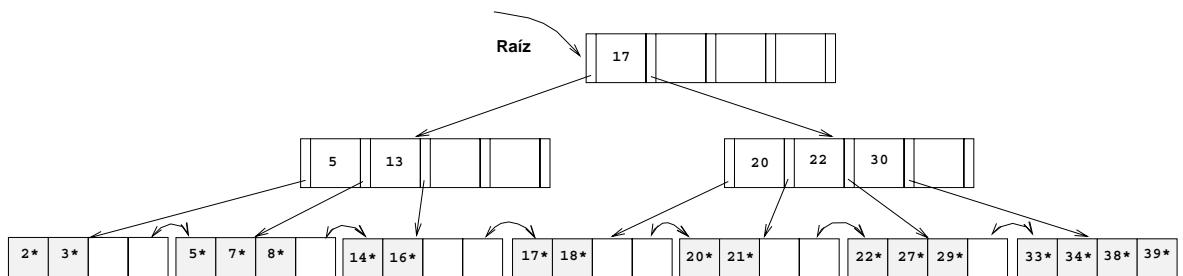


Figura 10.20 Árbol B+ después del borrado

³Es suficiente con trasladar solamente la entrada con valor de clave 20, pero se trasladan las dos para ilustrar lo que ocurre cuando se redistribuyen varias entradas.

Gestión de duplicados en sistemas comerciales. En un índice agrupado en Sybase ASE, las filas de datos se mantienen ordenadas en la página y en la colección de páginas de datos. Las páginas de datos están enlazadas bidireccionalmente en orden. Las filas con claves duplicadas se insertan en (o se eliminan del) conjunto ordenado de filas. Esto puede producir páginas de desbordamiento de filas con claves duplicadas que se insertan en la cadena de páginas, o páginas de desbordamiento vacías que se eliminan de la cadena de páginas. La inserción o el borrado de una clave duplicada no afecta los niveles de índice superiores a no ser que se produzca una división o una combinación de páginas que no son de desbordamiento. En IBM DB2, Oracle 8 y Microsoft SQL Server los duplicados se gestionan añadiendo un identificador de fila si es necesario eliminar valores de claves duplicados.

Para concluir este apartado obsérvese que se recupera solamente un hermano de un nodo. Si este nodo tiene entradas de más, se utiliza redistribución; si no, se combinan. Si el nodo tiene un segundo hermano, puede compensar recuperarlo también para comprobar si es posible la redistribución. Hay bastantes posibilidades de que la redistribución sea posible y, a diferencia de la combinación, no se propaga más allá del nodo padre. Además, las páginas disponen de más espacio, lo que reduce la probabilidad de una división debida a inserciones posteriores. (Recuérdese que los archivos normalmente crecen, no disminuyen.) Sin embargo, el número de veces que ocurre esto (el nodo está ocupado a menos de la mitad de su capacidad y el primer hermano no puede prestar una entrada) no es muy alto, así que no es necesario implementar este refinamiento del algoritmo básico que se ha presentado.

10.7 DUPLICADOS

Los algoritmos de búsqueda, inserción y borrado presentados ignoran el problema de las **claves duplicadas**, es decir, varias entradas de datos con el mismo valor de clave. A continuación se verá la forma de gestionar los duplicados.

El algoritmo de búsqueda básico supone que todas las entradas con un valor de clave determinado residen en una única página hoja. Una forma de satisfacer esta suposición es utilizando *páginas de desbordamiento* para tratar los duplicados. (En ISAM, por supuesto, existen páginas de desbordamiento en cualquier caso, y los duplicados se gestionan fácilmente.)

Habitualmente, sin embargo, se utiliza un enfoque diferente para los duplicados. Se gestionan exactamente igual a cualquier otra entrada y varias páginas hoja pueden contener entradas con un valor de clave determinado. Para recuperar todas las entradas de datos con un valor de clave dado, es necesario buscar la entrada de datos *más a la izquierda* con dicho valor de clave, y después recuperar posiblemente más de una página hoja (utilizando los punteros de secuencia de hojas). La modificación del algoritmo de búsqueda para encontrar la entrada de datos más a la izquierda en un índice con duplicados es un ejercicio interesante (de hecho, es el Ejercicio 10.11).

Un problema con este enfoque es que, cuando se elimina un registro, si se utiliza la alternativa (2) para las entradas de datos, puede ser ineficiente encontrar la entrada de datos

correspondiente, pues puede que se tengan que comprobar varias entradas $\langle \text{clave}, \text{idr} \rangle$ duplicadas con el mismo valor de *clave*. Este problema se puede resolver considerando el valor *idr* en la entrada de datos como *parte de la clave de búsqueda*, para posicionar la entrada de datos en el árbol. Esta solución de hecho convierte el índice en *único* (es decir, sin duplicados). Recuérdese que una clave de búsqueda puede ser cualquier secuencia de campos —en esta variante, el *idr* del registro de datos se trata fundamentalmente como otro campo al construir la clave de búsqueda—.

La alternativa (3) de entradas de datos lleva a una solución natural para los duplicados, pero si existe un gran número de duplicados, una sola entrada de datos podría ocupar múltiples páginas. Y por supuesto, cuando se elimina un registro de datos, encontrar el *idr* a eliminar puede ser ineficiente. La solución a este problema es similar a la comentada anteriormente para la alternativa (2): se puede mantener la lista de *idrs* en cada entrada de datos en orden (por ejemplo, por número de página y después por número de ranura si un *idr* está formado por un identificador de página y un identificador de ranura).

10.8 ÁRBOLES B+ EN LA PRÁCTICA

En este apartado se comentan diversos aspectos pragmáticos importantes.

10.8.1 Compresión de claves

La altura de un árbol B+ depende del *número de entradas de datos* y del *tamaño de las entradas de índice*. El tamaño de las entradas de índice determina el número de entradas de índice que caben en una página y, por consiguiente, el *grado de salida (fan-out)* del árbol. Como la altura del árbol es proporcional a $\log_{\text{grado de salida}}(\text{número de entradas de datos})$ y el número de operaciones de E/S en disco para recuperar una entrada de datos es igual a la altura (salvo que algunas páginas se encuentren en la memoria intermedia), es claramente importante maximizar el grado de salida para minimizar la altura.

Una entrada de índice contiene un valor de clave de búsqueda y un puntero de página. Por tanto el tamaño depende principalmente del tamaño del valor de la clave de búsqueda. Si éstos son muy largos (por ejemplo el nombre Devarakonda Venkataramana Sathyanarayana Seshasayee Yellamanchali Murthy, o Donaudampfschiffahrtsskapitänsanwärtersmütze), pocas entradas de índice entrarán en una página: el grado de salida será bajo, y la altura del árbol será grande.

Por otra parte, los valores de clave de búsqueda en las entradas de índice sólo se utilizan para dirigir el tráfico a la hoja apropiada. Cuando se quieren localizar entradas de datos con un valor de clave determinado, se compara este valor con los valores de las entradas de índice (en el camino desde la raíz a la hoja deseada). Durante la comparación en un nodo de nivel de índice, se quieren identificar dos entradas de índice con valores de clave k_1 and k_2 tales que el valor deseado k esté entre k_1 y k_2 . Para realizar esto, no es necesario almacenar los valores de clave enteros en las entradas de índice.

Por ejemplo, supóngase que hay dos entradas de índice adyacentes en un nodo, con valores de clave de búsqueda “Danilo Herrero” y “Devarakonda ...”. Para distinguir entre estos dos valores, basta con almacenar las formas abreviadas “Da” y “De”. De forma más general, el

Árboles B+ en sistemas reales. IBM DB2, Informix, Microsoft SQL Server, Oracle 8 y Sybase ASE soportan índices de árboles B+ agrupados y no agrupados, con algunas diferencias respecto a la forma de gestionar el borrado y los valores de clave duplicados. En Sybase ASE, dependiendo del esquema de control de concurrencia que se utilice para el índice, la fila borrada se elimina (combinando páginas si la ocupación de la página queda por debajo del umbral) o se marca como borrada; en este caso se utiliza un esquema de recolección de basura para recuperar espacio. En Oracle 8 se manejan los borrados marcando la fila como eliminada. Para recuperar el espacio ocupado por registros eliminados, se puede regenerar el índice de forma interactiva (es decir, mientras los usuarios continúan utilizando el índice) o unir páginas con baja ocupación (lo que no reduce la altura del árbol). La unión de páginas se realiza sobre el índice, mientras que la regeneración crea una copia. Informix gestiona los borrados marcando los registros como eliminados. DB2 y SQL Server eliminan los registros borrados y combinan las páginas cuando la ocupación es inferior al umbral.

Oracle 8 también permite que los registros de múltiples relaciones estén coagrupados en la misma página. La coagrupación puede estar basada en una clave de búsqueda de árbol B+ o en una asociación estática, y se pueden almacenar hasta 32 relaciones juntas.

significado de la entrada “Danilo Herrero” en el árbol B+ es que todos los valores del subárbol apuntado por el puntero a la izquierda de “Danilo Herrero” son menores que “Danilo Herrero”, y todos los valores en el subárbol apuntado por el puntero a la derecha de “Danilo Herrero” son (mayores o iguales a “Danilo Herrero” y) menores a “Devarakonda ...”.

Para asegurar que se preserva esta semántica en una entrada, mientras que se comprime la entrada con clave “Danilo Herrero”, debe examinarse el valor de clave más largo en el subárbol a la izquierda de “Danilo Herrero” y el valor de clave más corto en el subárbol a su derecha, no solamente las entradas de índice (“Dámaso León” y “Devarakonda ...”) que son sus vecinos. Esto se ilustra en la Figura 10.21; el valor “Daniel Juanes” es mayor a “Dan”, y por tanto, “Danilo Herrero” solo puede abbreviarse a “Dani”, en lugar de “Dan”.

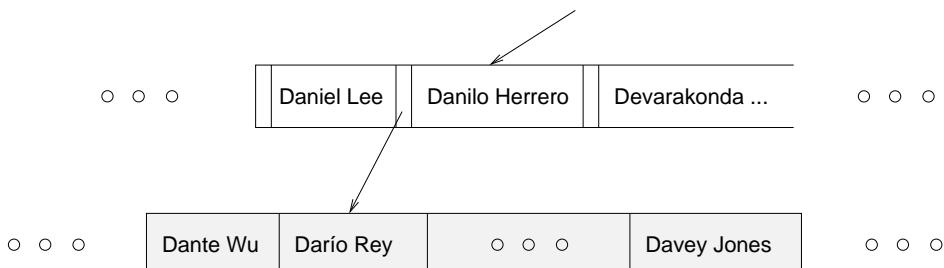


Figura 10.21 Ejemplo de compresión de claves de prefijo

Esta técnica, denominada **compresión de prefijo de claves** o simplemente **compresión de claves**, está soportada por muchas implementaciones comerciales de árboles B+, y puede

incrementar sustancialmente el grado de salida de un árbol. No se verán aquí los detalles de los algoritmos de inserción y borrado con compresión de claves.

10.8.2 Carga masiva de un árbol B+

Existen dos formas de añadir entradas a un árbol B+. En primer lugar, se puede disponer de una colección existente de registros de datos con un índice de árbol B+ sobre ella; siempre que se añade un registro a la colección, debe añadirse también la correspondiente entrada al árbol B+. (Por supuesto, se aplica lo mismo a los borrados.) En segundo lugar, se puede tener una colección de registros de datos para los que se desea crear un índice de árbol B+ sobre algún(os) campo(s) clave. En esta situación, se puede comenzar con un árbol vacío e insertar una entrada para cada registro de datos, una cada vez, utilizando el algoritmo de inserción estándar. Sin embargo, este enfoque es probablemente bastante costoso porque cada entrada requiere comenzar desde la raíz y descender hasta la página hoja correspondiente. Aunque las páginas de nivel de índice vayan a estar probablemente en la memoria intermedia en las distintas peticiones, la sobrecarga sigue siendo considerable.

Por este motivo muchos sistemas proporcionan una herramienta de *carga masiva* (*bulk-loading*) para crear un índice de árbol B+ sobre una colección existente de registros de datos. El primer paso es ordenar las entradas de datos $k*$ a insertar en el árbol B+ (que se va a crear) de acuerdo a la clave de búsqueda k . (Si las entradas son pares clave-puntero, su ordenación no implica la ordenación de los registros de datos a los que apuntan.) Se utilizará un ejemplo para ilustrar el algoritmo de carga masiva. Se supone que cada página de datos sólo puede contener dos entradas, y que cada página de índice puede contener dos entradas y un puntero adicional (es decir, se supone que el árbol B+ es de orden $d = 1$).

Después de ordenar las entradas de datos, se asigna una página vacía para la raíz y se inserta un puntero a la primera página de las entradas (ordenadas) en ella. Este proceso se muestra en la Figura 10.22, utilizando un conjunto de ejemplo de nueve páginas ordenadas de entradas de datos.

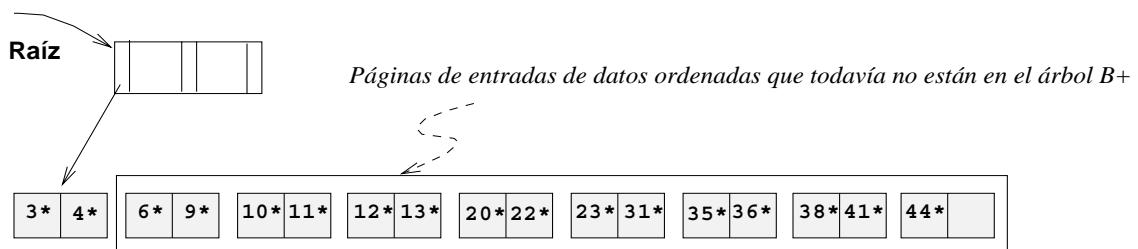


Figura 10.22 Paso inicial en la carga masiva de un árbol B+

Después se añade una entrada a la página raíz para cada página de datos ordenadas, de la forma \langle valor inferior de clave de la página, puntero a la página \rangle , hasta que se llene la página raíz; véase la Figura 10.23.

Para insertar la entrada de la siguiente página de datos, debe dividirse la raíz y crear una nueva página raíz. Este paso se muestra en la Figura 10.24.

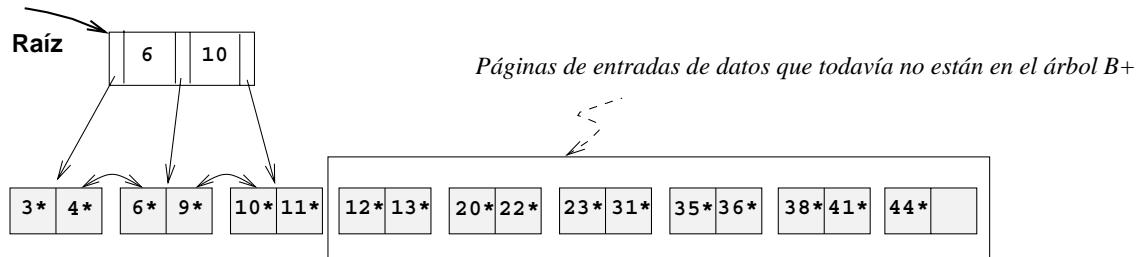


Figura 10.23 Llenado de la página raíz en la carga masiva de un árbol B+

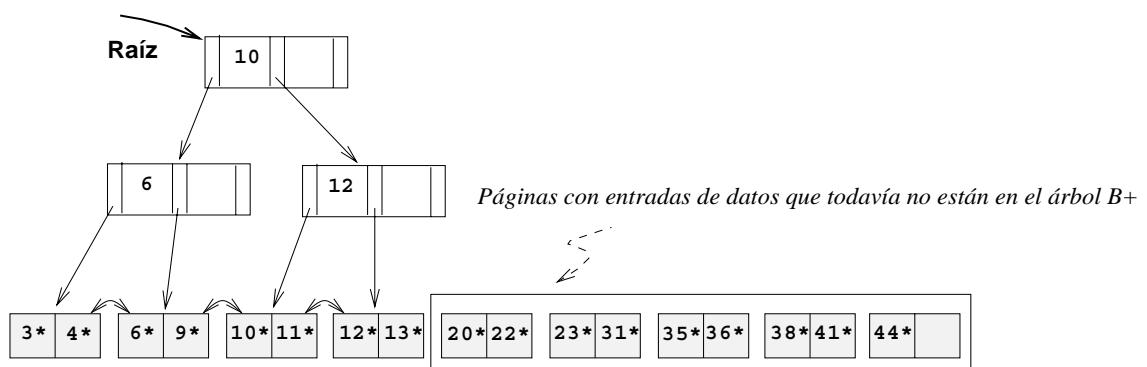


Figura 10.24 División de página durante la carga masiva de un árbol B+

Se redistribuyen uniformemente las entradas entre las dos hijas de la raíz, previendo el hecho de que el árbol B+ va a crecer. Aunque es difícil (!) ilustrar estas opciones cuando una página contiene dos entradas como máximo, se podrían haber dejado todas las entradas en la página antigua o bien llenarse una fracción de dicha página (por ejemplo, el 80 por 100). Estas alternativas son variantes simples de la idea básica.

Para continuar con el ejemplo de carga masiva, las entradas de páginas hoja siempre se insertan en la página índice más a la derecha que está justo encima del nivel de hojas. Cuando esta página se llena, se subdivide. Esta acción puede provocar la división de la página inmediatamente por encima, como se ilustra en las Figuras 10.25 y 10.26.

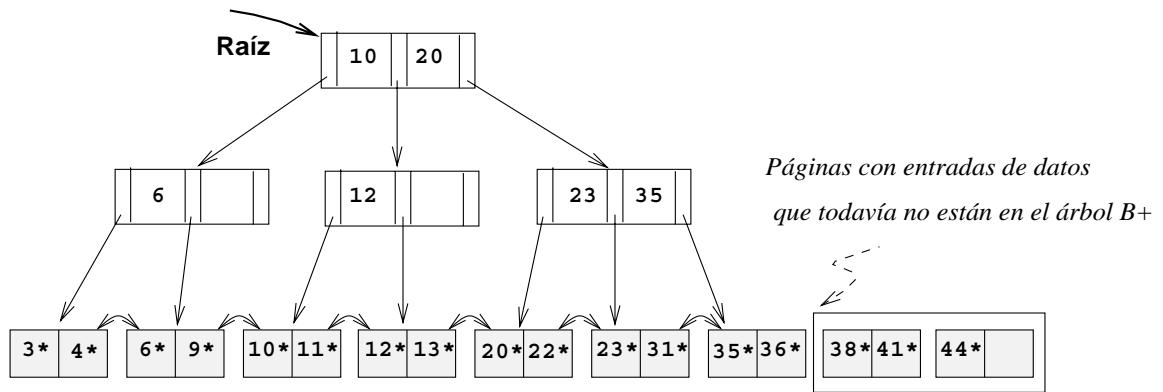


Figura 10.25 Antes de añadir una entrada para la página hoja que contiene 38*

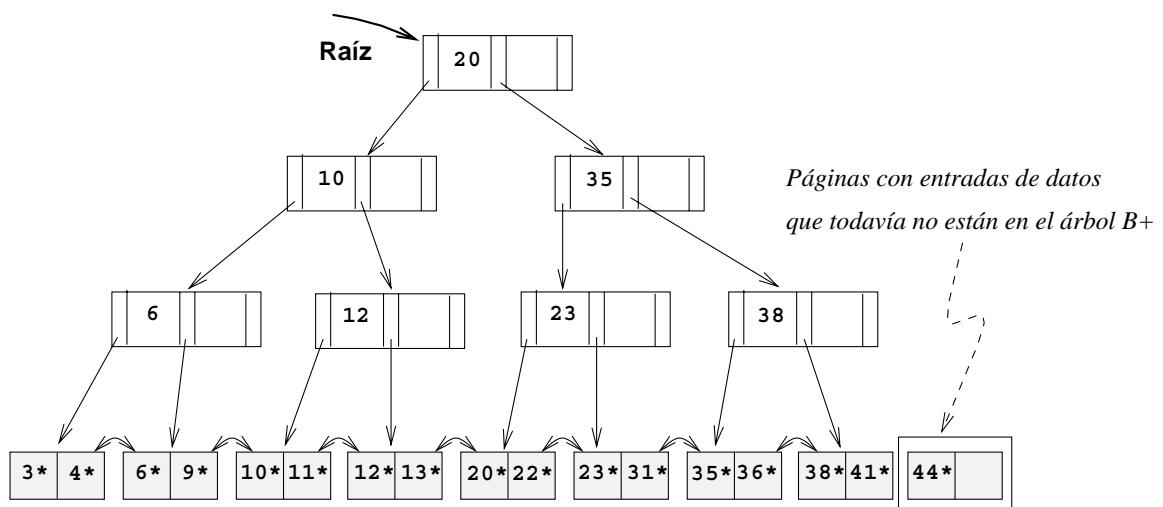


Figura 10.26 Despues de añadir una entrada para la página hoja que contiene 38*

Obsérvese que las divisiones sólo ocurren en el camino más a la derecha desde la raíz al nivel de las hojas. Se deja la finalización de este ejemplo como ejercicio.

Ahora se considerará el coste de crear un índice sobre una colección existente de registros. Esta operación está formada por tres pasos: (1) crear las entradas de datos a insertar en el índice (2) ordenarlas y (3) construir el índice a partir de las entradas ordenadas. El primer paso implica explorar los registros y escribir las entradas de datos correspondientes; el coste es $(R+E)$ operaciones de E/S, donde R es el número de páginas que contienen registros y E es el número de páginas que contienen entradas de datos. Las entradas de índice pueden ordenarse con un coste de alrededor de $3E$ operaciones de E/S. Estas entradas pueden insertarse después en el índice según son generadas, utilizando el algoritmo de carga masiva que se acaba de comentar. El coste del tercer paso, la inserción de entradas en el índice, es simplemente el coste de escribir todas las páginas de índice.

10.8.3 El concepto de orden

Los árboles B+ se han descrito utilizando el parámetro d para denotar la ocupación mínima. Es importante observar que el concepto de *orden* (es decir, el parámetro d), aunque es útil para enseñar los conceptos sobre árboles B+, normalmente debe relajarse en la práctica y sustituirse por un criterio de espacio físico; por ejemplo, que los nodos deben mantenerse por lo menos a la mitad de su capacidad.

Un motivo para esto es que los nodos hoja y los que no lo son normalmente pueden contener un número diferente de entradas. Recuérdese que los nodos son páginas en disco y los nodos que no son hojas sólo contienen claves de búsqueda y punteros a nodos, mientras que los nodos hoja pueden contener los registros de datos. Evidentemente, el tamaño de un registro de datos probablemente será más grande que el de una entrada de búsqueda, por lo que cabrán muchas más entradas de búsqueda que registros en una página de disco.

Otro motivo para relajar el concepto de orden es que la clave de búsqueda puede contener un campo de tipo cadena de caracteres (por ejemplo, el campo *nombre* de Alumnos) cuyo tamaño varíe de un registro a otro; este tipo de clave de búsqueda da lugar a entradas de datos y de índice de longitud variable, y el número de entradas que caben en una página se hace variable.

Finalmente, incluso si el índice se crea sobre un campo de tamaño fijo, puede haber varios registros con el mismo valor de la clave de búsqueda (por ejemplo, varios registros de Alumnos con el mismo valor de *nota* o *nombre*). Esta situación puede dar lugar también a entradas de hoja de tamaño variable (si se utiliza la alternativa (3) para las entradas de datos). Por todas estas complicaciones, el concepto de orden normalmente se sustituye por un criterio físico simple (por ejemplo, combinar si es posible cuando más de la mitad del espacio del nodo no se utiliza).

10.8.4 El efecto de las inserciones y borrados sobre los idr

Si las páginas hoja contienen registros de datos —es decir, el árbol B+ es un índice agrupado— entonces operaciones como divisiones, combinaciones y redistribuciones pueden cambiar los idr. Recuérdese que una representación típica de un idr es una combinación del número de página (físico) y el número de ranura. Este esquema permite trasladar registros dentro de una página si se elige un formato de página apropiado, pero no entre páginas, como es el

caso en operaciones tales como las divisiones de páginas. Así que a no ser que los idr sean independientes de los números de página, una división o una combinación en un árbol B+ agrupado puede requerir la actualización de otros índices sobre los mismos datos.

Un comentario similar se aplica para cualquier índice agrupado dinámico, independientemente de si está basado en árboles o en asociación. Por supuesto, este problema no se produce en índices no agrupados, puesto que sólo se trasladan las entradas de índice.

10.9 PREGUNTAS DE REPASO

Las respuestas a las preguntas de repaso se pueden encontrar en los apartados indicados.

- ¿Por qué los índices estructurados en árbol son buenos para búsquedas, y especialmente para selecciones por rangos? (**Apartado 10.1**)
- Describase el funcionamiento de las operaciones de búsqueda, inserción y borrado en índices ISAM. Analícese la necesidad de páginas de desbordamiento y su impacto potencial en el rendimiento. ¿A qué tipos de carga de trabajo son más vulnerables los índices ISAM, y qué tipos de carga de trabajo manejan bien? (**Apartado 10.2**)
- Las páginas hoja son las únicas afectadas en las actualizaciones de índices ISAM. Coméntense sus implicaciones para el bloqueo y el acceso concurrente. Compárese este aspecto en ISAM y en los árboles B+. (**Apartado 10.2.1**)
- ¿Cuáles son las diferencias principales entre los índices ISAM y los árboles B+? (**Apartado 10.3**)
- ¿Qué es el *orden* de un árbol B+? Describase el formato de los nodos en un árbol B+. ¿Por qué están enlazados los nodos en el nivel de las hojas? (**Apartado 10.3**)
- ¿Cuántos nodos deben examinarse en un árbol B+ para una búsqueda de igualdad? ¿Cuántos para una selección por rango? Compárense estos resultados con ISAM. (**Apartado 10.4**)
- Describase el algoritmo de inserción en un árbol B+, y explíquese cómo se eliminan las páginas de desbordamiento. ¿Bajo qué condiciones puede incrementar la altura del árbol una inserción? (**Apartado 10.5**)
- Durante el borrado, un nodo podría estar por debajo del umbral mínimo de ocupación. ¿Cómo puede gestionarse esto? ¿Bajo qué condiciones el borrado podría disminuir la altura del árbol? (**Apartado 10.6**)
- ¿Por qué las claves duplicadas requieren modificaciones en la implementación de las operaciones básicas sobre árboles B+? (**Apartado 10.7**)
- ¿Qué es la *compresión de claves* y por qué es importante? (**Apartado 10.8.1**)
- ¿Cómo puede construirse eficientemente un índice de árbol B+ para un conjunto de registros? Describase el algoritmo de *carga masiva*. (**Apartado 10.8.2**)
- Analícese el impacto de las divisiones de páginas en índices de árboles B+ agrupados. (**Apartado 10.8.4**)

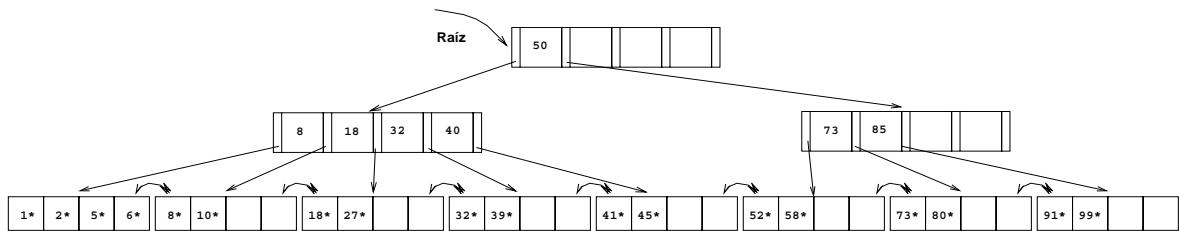


Figura 10.27 Árbol para el Ejercicio 10.1

EJERCICIOS

Ejercicio 10.1 Considérese el índice de árbol B+ de orden $d = 2$ mostrado en la Figura 10.27.

1. Muéstrese el árbol que resulta de insertar en este árbol una entrada de datos con clave 9.
2. Muéstrese el árbol que resulta de insertar en el árbol original una entrada de datos con clave 3. ¿Cuántas operaciones de lectura y de escritura de páginas requiere esta inserción?
3. Muéstrese el árbol que resulta de borrar la entrada de datos con clave 8 del árbol original, suponiendo que se comprueba si es posible la redistribución con el hermano izquierdo.
4. Muéstrese el árbol que resulta de borrar la entrada de datos con clave 8 del árbol original, suponiendo que se comprueba si es posible la redistribución con el hermano derecho.
5. Muéstrese el árbol que resulta de iniciar con el árbol original, insertar una entrada de datos con clave 46 y después borrar la entrada de datos con clave 52.
6. Muéstrese el árbol que resulta de borrar la entrada de datos con clave 91 del árbol original.
7. Muéstrese el árbol que resulta de iniciar con el árbol original, insertar una entrada de datos con clave 59 y después borrar la entrada de datos con clave 91.
8. Muéstrese el árbol que resulta de borrar sucesivamente las entradas de datos con claves 32, 39, 41, 45 y 73 del árbol original.

Ejercicio 10.2 Considérese el índice de árbol B+ mostrado en la Figura 10.28 que utiliza la alternativa (1) para las entradas de datos. Cada nodo intermedio puede contener hasta cinco punteros y cuatro valores de clave. Cada hoja puede contener hasta cuatro registros, y los nodos hoja están doblemente enlazados como es habitual, aunque estos enlaces no se muestran en la figura. Respóndase a las siguientes preguntas.

1. Nómbrense todos los nodos que deben leerse para contestar a la siguiente consulta: “Obtener todos los registros con clave de búsqueda mayor que 38.”
2. Muéstrese el árbol que resulta de insertar un registro en el árbol con clave de búsqueda 109.
3. Muéstrese el árbol que resulta de borrar el registro con clave de búsqueda 81 del árbol original.
4. Dígase un valor de clave tal que al insertarlo en el árbol (original) produzca un aumento de la altura del árbol.
5. Obsérvese que los subárboles A, B y C no están completamente especificados. No obstante, ¿qué puede inferirse del contenido y la forma de estos árboles?
6. ¿Cómo serían las respuestas a las preguntas precedentes si el índice fuera ISAM?
7. Supóngase que es un índice ISAM. ¿Cuál es el número mínimo de inserciones necesarias para crear una cadena de tres páginas de desbordamiento?

Ejercicio 10.3 Respóndase a las siguientes preguntas:

1. ¿Cuál es la utilización mínima de espacio de un índice en árbol B+?

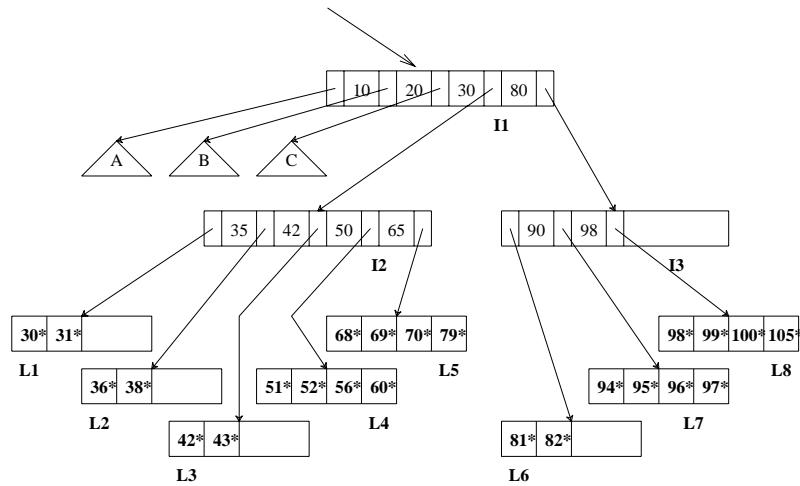


Figura 10.28 Árbol del Ejercicio 10.2

2. ¿Cuál es la utilización mínima de espacio de un índice ISAM?
3. Si un sistema de bases de datos soportara índices de árbol tanto estáticos como dinámicos (ISAM y árboles B+), ¿se podría considerar utilizar preferentemente un índice *estático* frente a uno *dinámico*?

Ejercicio 10.4 Supóngase que una página puede contener a lo sumo cuatro valores de datos y que todos ellos son números enteros. Utilizando solamente árboles B+ de orden 2, proporcionense ejemplos de cada uno de los siguientes:

1. Un árbol B+ cuya altura cambia de 2 a 3 cuando se inserta el valor 25. Muéstrese la estructura antes y después de la inserción.
2. Un árbol B+ en el que el borrado del valor 25 dé lugar a una redistribución. Muéstrese la estructura antes y después del borrado.
3. Un árbol B+ en el que el borrado del valor 25 produzca la combinación de dos nodos pero sin alterar la altura del árbol.
4. Una estructura ISAM con cuatro cajones, ninguno de los cuales tiene páginas de desbordamiento. Además, cada cajón tiene espacio para exactamente una entrada más. Muéstrese la estructura antes y después de insertar dos valores adicionales, elegidos de forma que se cree una página de desbordamiento.

Ejercicio 10.5 Considérese el árbol B+ mostrado en la Figura 10.29.

1. Identifíquese una lista de cinco entradas de datos tales que:
 - (a) Insertando las entradas en el orden mostrado y después borrándolas en orden inverso (por ejemplo, insertar *a*, insertar *b*, borrar *b*, borrar *a*) resulte el árbol original.
 - (b) Insertando las entradas en el orden mostrado y después borrándolas en orden inverso resulte un árbol distinto al original.
2. ¿Cuál es el número mínimo de inserciones de entradas de datos con claves distintas que producirá que la altura del árbol (original) cambie de su valor actual (altura 1) a 3?
3. ¿El número mínimo de inserciones anterior cambiaría si se pudieran insertar duplicados (varias entradas de datos con la misma clave), suponiendo que no se utilizan páginas de desbordamiento para gestionar duplicados?

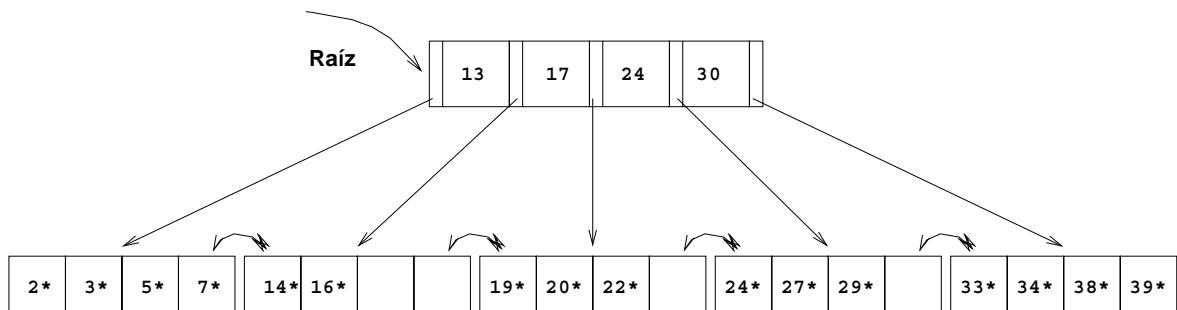


Figura 10.29 Árbol para el Ejercicio 10.5

Ejercicio 10.6 Respóndase al Ejercicio 10.5 suponiendo que el árbol es un árbol ISAM. (Algunos de los ejemplos pedidos pueden no existir —si es así, explíquese brevemente—.)

Ejercicio 10.7 Supóngase que se dispone de un archivo ordenado y se quiere construir un índice denso en árbol B+ primario sobre este archivo.

1. Una forma de realizar esto es explorando el archivo, registro por registro, insertando cada uno de ellos mediante el procedimiento de inserción en árboles B+. ¿Qué problemas de rendimiento y de utilización de almacenamiento produce este enfoque?
2. Explíquese la forma en que el algoritmo de carga masiva descrito en el texto mejora este esquema.

Ejercicio 10.8 Supóngase que se dispone de un índice en árbol B+ denso recién creado utilizando la alternativa (2) sobre un archivo de montículo que contiene 20.000 registros. El campo clave de este índice es una cadena de caracteres de 40 bytes, y es una clave candidata. Los punteros (es decir, identificadores de registro y de página) son valores de (a lo sumo) 10 bytes. El tamaño de una página en disco es de 1.000 bytes. El índice se creó de forma ascendente utilizando el algoritmo de carga masiva, y los nodos de cada nivel se han llenado tanto como era posible.

1. ¿Cuántos niveles tiene el árbol resultante?
2. Para cada nivel del árbol, ¿cuántos nodos hay?
3. ¿Cuántos niveles tendría el árbol resultante si se utilizara compresión de claves y esto redujera el tamaño medio de cada clave en una entrada a 10 bytes?
4. ¿Cuántos niveles tendría el árbol resultante sin compresión de claves pero con todas las páginas al 70 por 100 de ocupación?

Ejercicio 10.9 Los algoritmos de inserción y borrado en un árbol B+ se presentan como algoritmos recursivos. En el código de *insertar*, por ejemplo, se realiza una llamada al padre de un nodo N para insertar en (el subárbol cuya raíz es) el nodo N , y cuando termina esta llamada, el nodo actual es el padre de N . De este modo, no se mantienen “punteros al padre” en los nodos del árbol B+. Estos punteros no son parte de la estructura del árbol B+ por una buena razón, como muestra este ejercicio. Un enfoque alternativo que utiliza punteros al padre —de nuevo, recuérdese que tales punteros *no* son parte de la estructura estándar de un árbol B+!— en cada nodo parece ser más simple:

Buscar la hoja apropiada utilizando el algoritmo de búsqueda; después insertar la entrada y dividir si es necesario, propagando la división a los ancestros si es necesario (utilizando los punteros al padre para encontrarlo).

Considérese este enfoque alternativo (insatisfactorio):

1. Supóngase que un nodo interno N se divide en dos nodos N y N_2 . ¿Qué se puede decir sobre los punteros al padre en el hijo del nodo original N ?

<i>ide</i>	<i>nombre</i>	<i>usuario</i>	<i>edad</i>	<i>nota</i>
53831	Madayan	madayan@musica	11	1,8
53832	Guldu	guldu@musica	12	3,8
53666	Martín	martin@inf	18	3,4
53901	Martín	martin@juegos	18	3,4
53902	Martín	martin@fisica	18	3,4
53903	Martín	martin@ingles	18	3,4
53904	Martín	martin@genetica	18	3,4
53905	Martín	martin@astro	18	3,4
53906	Martín	martin@quimica	18	3,4
53902	Martín	martin@sanidad	18	3,8
53688	Rodríguez	rodriguez@ie	19	3,2
53650	Rodríguez	rodriguez@mat	19	3,8
54001	Rodríguez	rodriguez@ie	19	3,5
54005	Rodríguez	rodriguez@inf	19	3,8
54009	Rodríguez	rodriguez@astro	19	2,2

Figura 10.30 Un ejemplar de la relación Alumnos

2. Sugíéranse dos formas de tratar los punteros al padre inconsistentes en el hijo del nodo N .
3. Para cada una de estas sugerencias, identifíquese una ventaja potencial (importante).
4. ¿Qué conclusiones se pueden obtener de este ejercicio?

Ejercicio 10.10 Considérese el ejemplar de la relación Alumnos mostrado en la Figura 10.30. Muéstrese un árbol B+ de orden 2 en cada uno de los casos siguientes, suponiendo que los duplicados se gestionan mediante páginas de desbordamiento. Indíquense claramente las entradas de datos (es decir, no debe usarse el convenio $k*$).

1. Un índice en árbol B+ sobre *edad* utilizando la alternativa (1) para las entradas de datos.
2. Un índice denso de árbol B+ sobre *nota* utilizando la alternativa (2) para las entradas de datos. Para esta pregunta, supóngase que estas tuplas se almacenan en un archivo ordenado como se muestra en la Figura 10.30: la primera tupla está en la página 1, ranura 1; la segunda en la página 1, ranura 2; y así sucesivamente. Cada página puede almacenar hasta tres registros de datos. Puede usarse $\langle id\text{-página}, ranura \rangle$ para identificar una tupla.

Ejercicio 10.11 Supóngase que los duplicados se gestionan utilizando el enfoque sin páginas de desbordamiento descrito en el Apartado 10.7. Describáse un algoritmo para buscar la ocurrencia más a la izquierda de una entrada de datos con valor de la clave de búsqueda K .

Ejercicio 10.12 Respóndase al Ejercicio 10.10 suponiendo que los duplicados se gestionan sin utilizar páginas de desbordamiento usando el enfoque alternativo sugerido en el Apartado 9.7.



EJERCICIOS BASADOS EN PROYECTOS

Ejercicio 10.13 Compárense las interfaces públicas de archivos de montículo, índices de árbol B+ e índices asociativos lineales. ¿Cuáles son las similitudes y diferencias? Explíquese por qué existen estas similitudes y diferencias.

Ejercicio 10.14 Este ejercicio implica el uso de Minibase para explorar más los ejercicios anteriores (no de proyecto).

1. Creéñense los árboles mostrados en los ejercicios anteriores y visualíicense utilizando el visualizador de árboles B+ de Minibase.
2. Verifíquense las respuestas a los ejercicios que requieran inserción y borrado de entradas de datos haciéndolas en Minibase y observando los árboles resultantes con el visualizador.

Ejercicio 10.15 (*Nota para los profesores: si se propone este ejercicio deben proporcionarse detalles adicionales; véase la página Web del libro original.*) Impleméntense árboles B+ sobre el código de bajo nivel de Minibase.

NOTAS BIBLIOGRÁFICAS

La versión original de los árboles B+ fue presentada por Bayer y McCreight [47]. Los árboles B+ se describen en [281] y [126]. Los índices de árboles B para distribuciones de datos sesgadas se estudian en [169]. La estructura de indexación VSAM se describe en [465]. En [51] se revisan varias estructuras de árbol para soportar consultas por rangos. Un artículo pionero en claves de búsqueda multatributo es [307].



11

INDEXACIÓN BASADA EN ASOCIACIÓN

- ☞ ¿Cuál es la intuición tras los índices estructurados por asociación? ¿Por qué son especialmente buenos para búsquedas por igualdad, pero inútiles en las selecciones por rango?
- ☞ ¿Qué es la asociación extensible? ¿Cómo gestiona la búsqueda, la inserción y el borrado?
- ☞ ¿Qué es la asociación lineal? ¿Cómo gestiona la búsqueda, la inserción y el borrado?
- ☞ ¿Cuáles son las similitudes y diferencias entre la asociación extensible y la lineal?
- ➡ **Conceptos clave:** función de asociación, cajón, páginas primarias y de desbordamiento, índices asociativos estáticos y dinámicos; asociación extensible, directorio de cajones, división de un cajón, profundidad global y local, duplicación de directorios, colisiones y páginas de desbordamiento; asociación lineal, serie de divisiones, familia de funciones de asociación, páginas de desbordamiento, elección del cajón a dividir y tiempo de división; relación entre directorio de asociación extensible y familia de funciones de asociación de la asociación lineal, necesidad de páginas de desbordamiento en ambos esquemas en la práctica, uso de un directorio para la asociación lineal.

No es como el caos, mezclado y majado,
sino como el mundo, confundido por su armonía:
donde se revela orden en la variedad.

— Alexander Pope, *El bosque de Windsor*

En este capítulo se verán organizaciones de archivo que son excelentes para las selecciones por igualdad. La idea básica es utilizar una *función de asociación* que hace corresponder valores de un campo de búsqueda en un rango de *números de cajón* para encontrar la página a la

que pertenece la entrada de datos solicitada. Se utilizará un esquema sencillo denominado *asociación estática* para introducir la idea. Este esquema, como ISAM, tiene el problema de las largas cadenas de desbordamiento, que pueden afectar al rendimiento. Se presentan dos soluciones a este problema. El esquema de *asociación extensible* utiliza un directorio para realizar inserciones y borrados eficientemente sin páginas de desbordamiento. El esquema de *asociación lineal* utiliza una política inteligente para la creación de nuevos cajones y permite inserciones y borrados eficientes sin el uso de un directorio. Aunque se utilizan páginas de desbordamiento, la longitud de las cadenas de desbordamiento raramente es mayor que dos.

Las técnicas de indexación basadas en asociación no admiten búsquedas por rangos, desafortunadamente. Las técnicas basadas en árboles, tratadas en el Capítulo 10, pueden sopor tarlas eficientemente y son casi tan rápidas como la indexación asociativa para las selecciones por igualdad. De esta forma, muchos sistemas comerciales sólo soportan índices basados en árboles. Sin embargo, las técnicas asociativas demuestran ser muy útiles implementando operaciones relacionales tales como uniones. En particular, el método de unión de bucles anidados de índice genera muchas consultas por igualdad, y la diferencia respecto al coste entre un índice asociativo y uno basado en árbol puede ser significativa en este contexto.

El resto de este capítulo está organizado de la siguiente forma. En el Apartado 11.1 se presenta la asociación estática. Como en el caso de ISAM, su inconveniente es que el rendimiento se degrada a medida que el número de datos crece y disminuye. Se tratará una técnica de asociación dinámica, denominada *asociación extensible*, en el Apartado 11.2, y otra técnica dinámica, llamada *asociación lineal*, en el Apartado 11.3. Se compararán ambas técnicas en el Apartado 11.4.

11.1 ASOCIACIÓN ESTÁTICA

El esquema de asociación estática se ilustra en la Figura 11.1. Las páginas que contienen los datos pueden verse como una colección de **cajones**, con una página **primaria** y posiblemente páginas de **desbordamiento** adicionales por cada cajón. Un archivo está inicialmente formado por cajones 0 a $N - 1$, con una página primaria por cajón. Los cajones contienen **entradas de datos**, que pueden ser de cualquiera de las tres alternativas comentadas en el Capítulo 9.

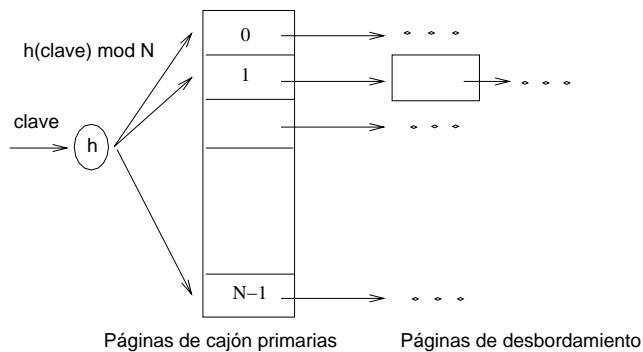


Figura 11.1 Asociación estática

Para buscar una entrada de datos se aplica una **función de asociación** h para identificar el cajón al que pertenece y después buscar en ese cajón. Para acelerar la búsqueda de un cajón, se pueden mantener las entradas de datos ordenadas por el valor de la clave de búsqueda; en este capítulo no se ordenan las entradas, y el orden en el que aparecen en el cajón no es significativo. Para insertar una entrada de datos se utiliza la función de asociación para identificar el cajón correcto y se introduce la entrada de datos en él. Si no hay espacio para esta entrada, se asigna una nueva página de *desbordamiento*, se introduce la entrada de datos en esta página y se añade la página a la **cadena de desbordamiento** del cajón. Para borrar una entrada de datos, se utiliza la función de asociación para identificar el cajón correcto, se localiza la entrada de datos buscándola en el cajón y se elimina. Si esta entrada es la última de una página de desbordamiento, se elimina la página de la cadena de desbordamiento y se añade a una lista de *páginas libres*.

La función de asociación es un componente importante del enfoque asociativo. Debe distribuir uniformemente valores del dominio del campo de búsqueda sobre el conjunto de cajones. Si se dispone de N cajones, numerados del 0 al $N - 1$, una función de asociación h de la forma $h(\text{valor}) = (a * \text{valor} + b)$ funciona bien en la práctica. (El cajón identificado es $h(\text{valor}) \bmod N$.) Las constantes a y b pueden elegirse para “ajustar” la función asociativa.

Como se conoce el número de cajones en un archivo asociativo estático cuando se crea, las páginas primarias pueden almacenarse en páginas consecutivas en disco. Por lo tanto, una búsqueda requiere idealmente sólo una operación de E/S, y las operaciones de inserción y borrado requieren dos (leer y escribir la página) aunque el coste podría ser mayor si hubiera páginas de desbordamiento. A medida que crece el archivo, se pueden desarrollar largas cadenas de desbordamiento. Como buscar un cajón requiere buscar (en general) en todas las páginas de su cadena de desbordamiento, es fácil ver el deterioro del rendimiento. Manteniendo inicialmente páginas llenas al 80 por 100, se pueden evitar páginas de desbordamiento si el archivo no crece demasiado, pero en general la única forma de deshacerse de las cadenas de desbordamiento es creando un nuevo archivo con más cajones.

El problema principal de la asociación estática es que el número de cajones es fijo. Si un archivo disminuye mucho su tamaño, se desperdicia gran cantidad de espacio; más importante, si crece mucho, se desarrollan largas cadenas de desbordamiento, lo que se traduce en un rendimiento pobre. Por tanto, la asociación estática se puede comparar a la estructura ISAM (Apartado 10.2) que también puede desarrollar largas cadenas de desbordamiento cuando se producen inserciones en la misma hoja. La asociación estática también tiene las mismas ventajas que ISAM respecto a los accesos concurrentes (véase el Apartado 10.2.1).

Una alternativa sencilla para la asociación estática consiste en “reasociar” periódicamente el archivo para restaurar la situación ideal (sin páginas de desbordamiento y con una ocupación de aproximadamente el 80 por 100). No obstante, la reasociación lleva tiempo y no se puede utilizar el índice mientras se está realizando. Otra alternativa es el uso de técnicas de **asociación dinámica** tales como la asociación extensible o lineal, que tratan de forma elegante las inserciones y los borrados. Se considerarán estas técnicas en el resto de este capítulo.

11.1.1 Notación y convenios

En el resto de este capítulo se utilizarán los siguientes convenios. Como en el capítulo anterior, en los registros con clave de búsqueda k se denota la entrada de datos de índice con $k*$. En los índices asociativos, el primer paso en la búsqueda, inserción o borrado de una entrada de datos con clave de búsqueda k consiste en aplicar una función de asociación h a k ; se denota esta operación con $h(k)$, y el valor $h(k)$ identifica el cajón de la entrada de datos $k*$. Obsérvese que dos claves de búsqueda diferentes pueden tener el mismo valor de asociación.

11.2 ASOCIACIÓN EXTENSIBLE

Para comprender la asociación extensible se comenzará considerando un archivo asociativo estático. Si se tiene que insertar una nueva entrada de datos en un cajón lleno, es necesario añadir una página de desbordamiento. Si no se quieren añadir páginas de desbordamiento, una posible solución consiste en reorganizar el archivo en ese momento duplicando el número de cajones y distribuyendo las entradas en este nuevo conjunto de cajones. Esta solución tiene un defecto importante —tiene que leerse el archivo entero, y tienen que escribirse el doble de páginas para realizar la reorganización—. Sin embargo, este problema puede superarse con una idea sencilla: utilizar un **directorio** de punteros a cajones, y duplicar sólo el tamaño del directorio y dividir *solamente* el cajón que se ha desbordado.

Para comprender esta idea considérese el archivo de ejemplo mostrado en la Figura 11.2. El directorio está formado por un vector de cuatro elementos, cada uno de los cuales es un puntero a un cajón. (Los campos *profundidad global* y *profundidad local* se comentarán más adelante, y se pueden ignorar por ahora.) Para localizar una entrada de datos se aplica una función de asociación (*hash*) al campo de búsqueda y se toman los últimos 2 bits de su representación binaria para obtener un número entre 0 y 3. El puntero de esta posición del vector proporciona el cajón que corresponde; se supone que cada cajón puede contener cuatro entradas de datos. Por consiguiente, para localizar una entrada de datos con un valor de asociación 5 (101 en binario) se obtiene el elemento del directorio 1 y se sigue el puntero a la página de datos (cajón B en la figura).

Para insertar una entrada de datos se busca el cajón correspondiente. Por ejemplo, para insertar una entrada de datos con un valor de asociación 13 (representado como 13*) se examina el elemento del directorio 01 y se va a la página que contiene las entradas de datos 1*, 5* y 21*. Como la página tiene espacio para una entrada de datos adicional, basta con insertar la entrada (Figura 11.3).

A continuación se considerará la inserción de una entrada de datos en un cajón que esté lleno. La esencia de la asociación extendida reside en la forma en que se trata este caso. Considérese la inserción de la entrada de datos 20* (10100 en binario). El elemento de directorio 00 lleva al cajón A, que ya está lleno. Por tanto debe **dividirse** primero el cajón asignando un nuevo cajón¹ y redistribuir el contenido (incluyendo la nueva entrada a insertar) entre el cajón antiguo y su “imagen dividida”. Para redistribuir entradas entre ambos, se consideran los últimos tres bits de $h(r)$; los últimos dos bits son 00, indicando una

¹Como no hay páginas de desbordamiento en la asociación extensible, un cajón puede verse como una única página.

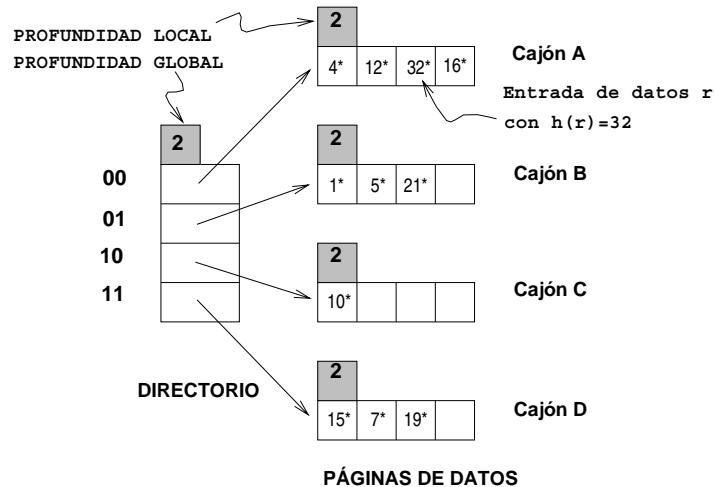


Figura 11.2 Ejemplo de archivo asociativo extensible

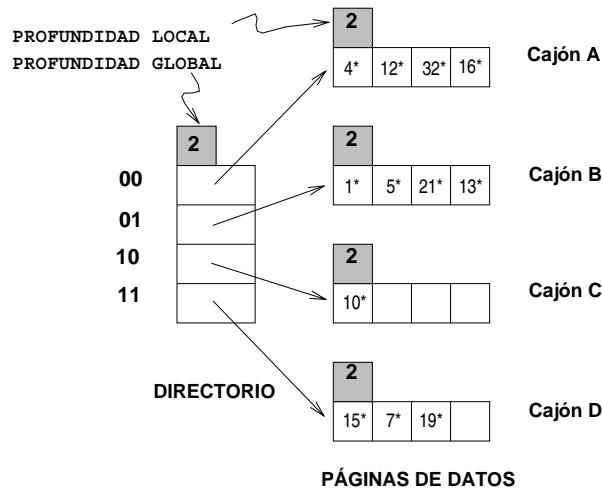


Figura 11.3 Despues de insertar la entrada r con $h(r) = 13$

entrada de datos que pertenece a uno de estos dos cajones, y el tercer bit discrimina entre ellos. La redistribución de entradas se ilustra en la Figura 11.4.

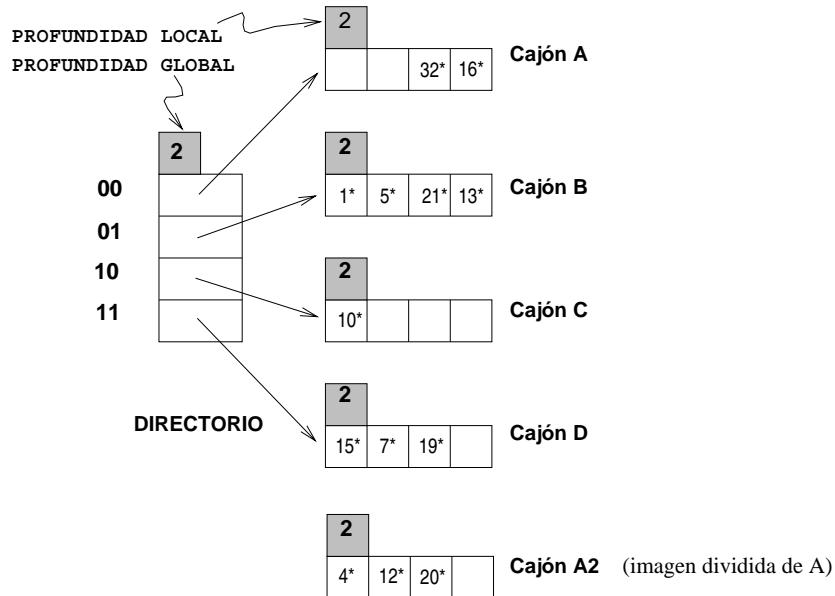


Figura 11.4 Durante la inserción de la entrada r con $h(r)=20$

Ahora obsérvese un problema que debe resolverse —se necesitan tres bits para discriminar entre dos de las páginas de datos (A y A2) pero el directorio sólo tiene suficiente espacio para almacenar los patrones de dos bits—. La solución consiste en *duplicar el directorio*. Los elementos que se diferencian solamente en el tercer bit comenzando por el final se dice que “corresponden”: los *elementos que corresponden* en el punto del directorio al mismo cajón con la excepción de los elementos que corresponden al cajón dividido. En el ejemplo, el cajón 0 fue dividido; de este modo, el nuevo elemento de directorio 000 apunta a una de las versiones divididas y el nuevo elemento 100 apunta a la otra. El archivo de ejemplo después de completar todos los pasos en la inserción de 20* se muestra en la Figura 11.5.

Por tanto, duplicar el archivo requiere la creación de una nueva página de cajón, escribir tanto ésta como el cajón que se divide, y duplicar el vector del directorio. El directorio probablemente es mucho más pequeño que el archivo porque cada elemento es solamente un identificador de página, y puede duplicarse simplemente copiándolo (y ajustando los elementos de los cajones divididos). El coste de duplicarlo es perfectamente aceptable.

Se puede observar que la técnica básica utilizada en la asociación extensible consiste en tratar el resultado de aplicar una función de asociación h como un número binario e interpretar los últimos d bits, donde d depende del tamaño del directorio, como un desplazamiento dentro del directorio. En el ejemplo anterior, d es originalmente 2 porque solo había cuatro cajones; después de la división, d pasa a ser 3 porque ahora se dispone de ocho cajones. Un corolario es que, cuando se distribuyen entradas entre un cajón y su imagen dividida, se debería hacer sobre la base del d -ésimo bit. (Obsérvese la forma en que las entradas se redistribuyen en el ejemplo; véase la Figura 11.5.) El número d , denominado **profundidad global** del archivo

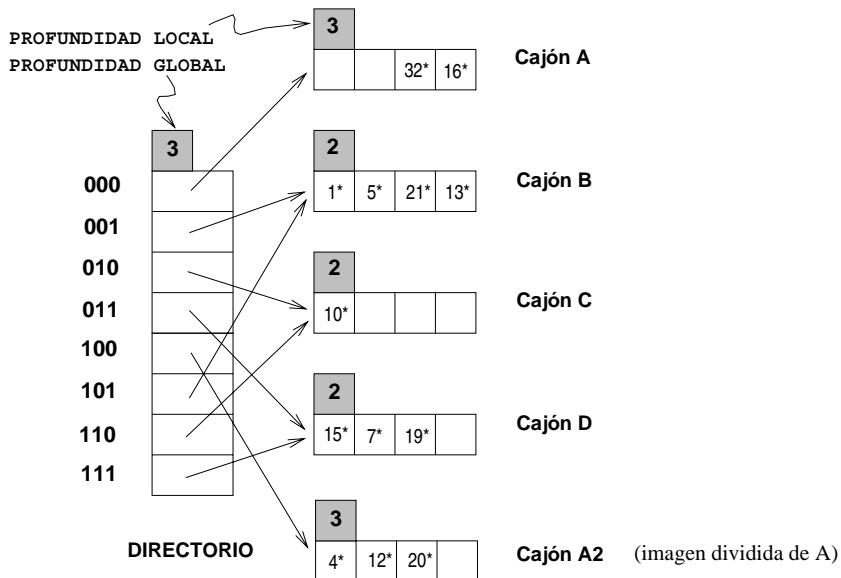


Figura 11.5 Despues de insertar la entrada r con $h(r) = 20$

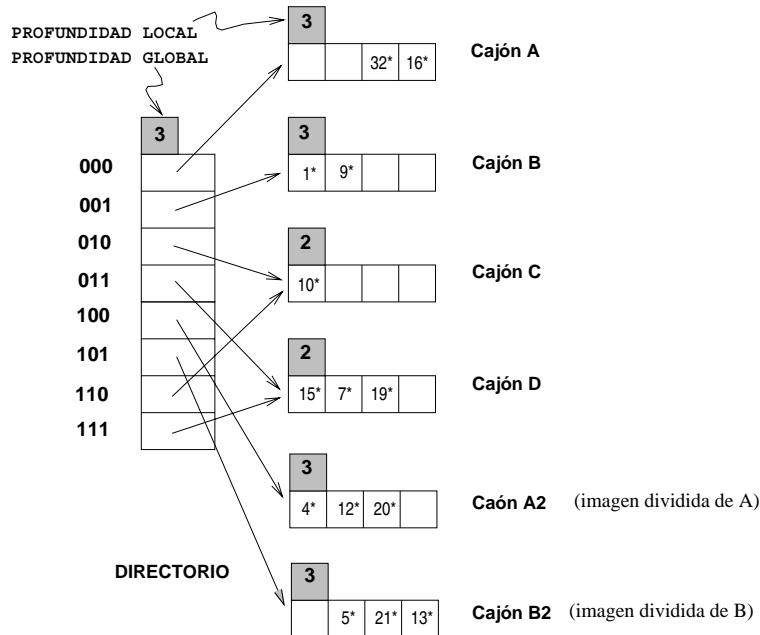
asociativo, se mantiene como parte de la cabecera del archivo. Se utiliza cada vez que se necesita localizar una entrada de datos.

Un punto importante es si la división de un cajón necesita la duplicación de un directorio. Considérese el ejemplo anterior, como se muestra en la Figura 11.5. Si se inserta 9^* , pertenece al cajón B; este cajón ya está lleno. Se puede tratar esta situación dividiendo el cajón y utilizando los elementos del directorio 001 y 101 para apuntar al cajón y su imagen dividida, como se muestra en la Figura 11.6.

Por lo tanto, una división de cajón no requiere necesariamente una duplicación de directorio. Sin embargo, si cualquiera de los cajones A o A2 se llenan y una inserción fuerza una división de cajón, entonces es obligatorio duplicar el directorio de nuevo.

Para diferenciar entre estos casos y determinar si se necesita duplicar el directorio, se mantiene una **profundidad local** para cada cajón. Si se divide un cajón cuya profundidad local es igual a la profundidad global, el directorio debe duplicarse. Volviendo al ejemplo anterior, cuando se inserta 9^* en el índice mostrado en la Figura 11.5, pertenece al cajón B con profundidad local 2, mientras que la profundidad global era 3. Aunque el cajón se dividiera, el directorio no tendría que duplicarse. Los cajones A y A2, por otra parte, tienen profundidad local igual a la global y, si crecen hasta llenarse y se dividen, el directorio debe entonces duplicarse.

Inicialmente todas las profundidades locales son iguales a la profundidad global (que es el número de bits necesarios para expresar el número total de cajones). Se incrementa en uno la profundidad global cada vez que el directorio se duplica. También, siempre que se divide un cajón (tanto si la división lleva a duplicar el directorio como si no) se incrementa en uno la profundidad local del cajón dividido y se asigna esta misma profundidad local (incrementada) a su imagen (recién creada). Intuitivamente, si un cajón tiene una profundidad local l , los

Figura 11.6 Despues de insertar la entrada r con $h(r) = 9$

valores de asociación de las entradas de datos tienen los últimos l bits iguales; más aún, ninguna entrada de datos de ningún otro cajón del archivo tiene un valor de asociación con los últimos l bits iguales. Un total de 2^{d-l} elementos de directorio apuntan a un cajón con profundidad local l ; si $d = 1$, exactamente un elemento de directorio apunta al cajón y la división de ese cajón requiere duplicar el directorio.

Una observación final es que también se pueden usar los primeros d bits (los bits *más significativos*) en lugar de los últimos (los *menos significativos*) pero en la práctica se utilizan los *últimos* d bits. La razón es que un directorio puede duplicarse simplemente copiándolo.

En resumen, una entrada de datos puede localizarse calculando su valor de asociación, tomando los últimos d bits y examinando el cajón apuntado por este elemento de directorio. Para las inserciones, la entrada de datos se coloca en el cajón al que pertenece y el cajón se divide si es necesario añadir espacio. Un cajón dividido lleva a incrementar la profundidad local y, si ésta se hace mayor que la profundidad global, lleva también a duplicar el directorio (e incrementar la profundidad global).

Para los borrados se localiza la entrada de datos y se elimina. Si el borrado deja el cajón vacío, puede combinarse con su imagen dividida, aunque este paso se omite a menudo en la práctica. Combinar cajones disminuye la profundidad local. Si cada elemento de directorio apunta al mismo cajón que su imagen dividida (es decir, 0 y 2^{d-1} apuntan al mismo cajón A; 1 y $2^{d-1} + 1$ apuntan al mismo cajón B, que puede o no ser idéntico a A; etc.) se puede reducir a la mitad el directorio y la profundidad global, aunque este paso no es necesario para ser correcto.

Los ejemplos de inserción pueden desarrollarse en orden inverso como ejemplos de borrado. (Comenzando con la estructura mostrada después de una inserción y borrando el elemento insertado. En cada caso el resultado debería ser la estructura original.)

Si el directorio cabe en memoria, una selección por igualdad puede contestarse con un único acceso a disco, como en la asociación estática (en ausencia de páginas de desbordamiento) pero en caso contrario se necesitan dos operaciones de E/S. Como ejemplo típico, un archivo de 100MB con 100 bytes por entrada de datos y un tamaño de página de 4KB contiene un millón de entradas de datos y sólo 25.000 elementos en el directorio. (Cada página/cajón contiene alrededor de 40 entradas de datos, y solamente hay un elemento de directorio por cajón.) De este modo, aunque las selecciones por igualdad pueden ser el doble de lentas que para los archivos asociativos estáticos, lo más probable es que el directorio quepa en memoria y el rendimiento sea el mismo que para los archivos asociativos estáticos.

Por otra parte, el directorio crece a rachas, y puede hacerse grande para *distribuciones de datos sesgadas* (en las que no es válida la suposición de que las páginas de datos contienen aproximadamente el mismo número de entradas de datos). En el contexto de los archivos asociativos, en una **distribución de datos sesgada** la distribución de los *valores de asociación de los campos de búsqueda* (más que las distribuciones de los propios valores de los campos) está sesgada (en altibajos, no uniformemente). Incluso si la distribución de los valores de búsqueda está sesgada, la elección de una buena función de asociación normalmente da lugar a una distribución uniforme de los valores de asociación; el sesgo no es por lo tanto un problema en la práctica.

Más aún, las **colisiones** (entradas de datos con el mismo valor de asociación) pueden causar un problema y deben gestionarse especialmente: cuando hay más entradas de datos con el mismo valor de asociación de las que caben en una página, es necesario utilizar páginas de desbordamiento.

11.3 ASOCIACIÓN LINEAL

La asociación lineal es una técnica de asociación dinámica, al igual que la asociación extensible, que se ajusta elegantemente a las inserciones y los borrados. A diferencia de la asociación extensible, no requiere de un directorio, trata de forma natural las colisiones y ofrece mucha flexibilidad respecto al tiempo utilizado para la división de cajones (permitiendo equilibrar cadenas de desbordamiento ligeramente mayores con una mayor utilización media del espacio). Si la distribución de datos es muy sesgada, no obstante, las cadenas de desbordamiento pueden hacer que el rendimiento de la asociación lineal sea peor que la extensible.

El esquema utiliza una *familia* de funciones de asociación h_0, h_1, h_2, \dots , con la propiedad de que el rango de cada cajón es el doble de su predecesor. Es decir, si h_i hace corresponder una entrada de datos con uno de los M cajones, h_{i+1} hace corresponder una entrada de datos con uno de $2M$ cajones. Esta familia normalmente se obtiene eligiendo una función de asociación h y un número inicial N de cajones², y definiendo $h_i(\text{valor}) = h(\text{valor}) \bmod (2^i N)$. Si se elige un N potencia de 2, entonces se aplica h y se obtienen los últimos d_i bits; d_0 es el número de bits necesarios para representar N , y $d_i = d_0 + i$. Normalmente se elige que h sea una función que haga corresponder una entrada de datos a algún número entero. Supóngase

²Obsérvese que 0 a $N - 1$ no es el rango de h !

que se asigna 32 como el número inicial N . En este caso d_0 es 5, y h_0 es por tanto $h \bmod 32$, es decir, un número en el rango 0 a 31. El valor de d_1 es $d_0+1 = 6$, y h_1 es $h \bmod (2*32)$, es decir, un número en el rango 0 a 63. h_2 produce un número en el rango 0 a 127, y así sucesivamente.

La idea se entiende mejor en términos de **rondas** de divisiones. Durante la ronda número $Nivel$, sólo están en uso las funciones de asociación h_{Nivel} y $h_{Nivel+1}$. De este modo, los cajones del archivo que se dividen al inicio de la ronda, uno por uno del primer cajón al último, duplicando por tanto el número de cajones. En cualquier punto dentro de una ronda, por tanto, se dispone de cajones que se han dividido, cajones que todavía van a dividirse, y cajones creados por divisiones en esta ronda, como se ilustra en la Figura 11.7.

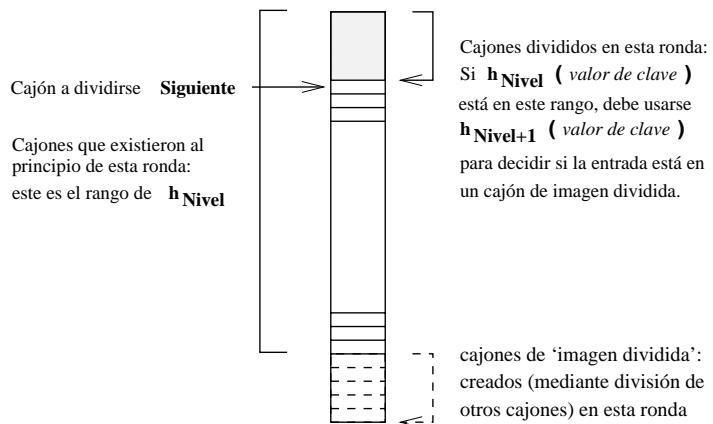
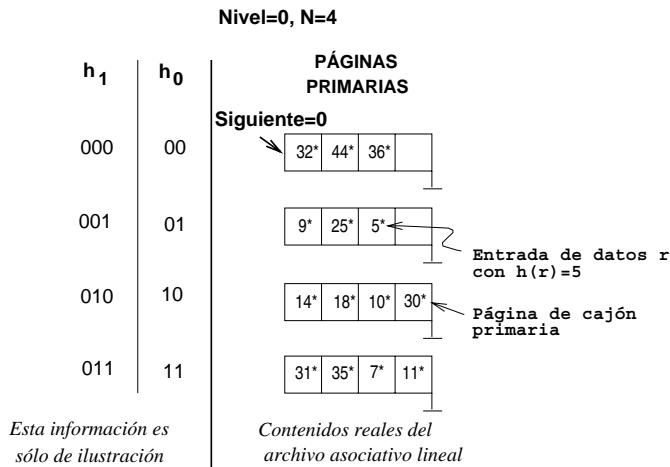


Figura 11.7 Cajones durante una ronda en asociación lineal

Considérese ahora la búsqueda de una entrada de datos con un valor de clave de búsqueda determinado. Se aplica la función de asociación h_{Nivel} , y si ésta lleva a uno de los cajones no divididos, basta con buscar en él. Si lleva a uno de los cajones divididos, la entrada puede estar en él o haberse trasladado al nuevo cajón creado anteriormente en esta ronda cuando se dividió este cajón; para determinar cuál de los cajones contiene la entrada se aplica $h_{Nivel+1}$.

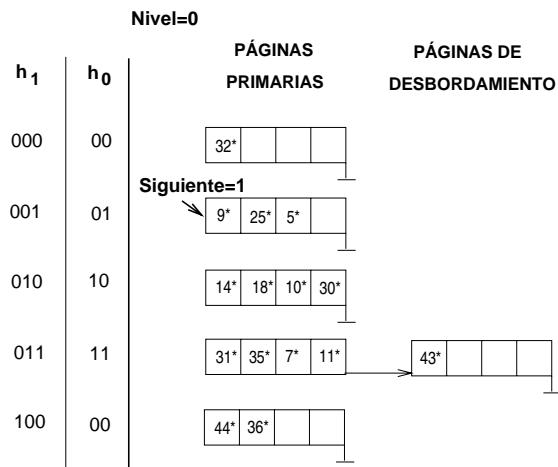
A diferencia de la asociación extensible, cuando una inserción desencadena una división, el cajón en el que se inserta la entrada de datos no es necesariamente el cajón que se divide. Se añade una página de desbordamiento para almacenar la entrada que se acaba de insertar (que desencadenó la división), como en la asociación estática. Sin embargo, como el cajón a dividir se elige mediante turno rotatorio (*round-robin*), finalmente todos los cajones se dividen, redistribuyendo de este modo las entradas de datos en cadenas de desbordamiento antes de que éstas tengan más de dos páginas.

Ahora se describirá la asociación lineal en más detalle. Se utiliza un contador $Nivel$ para indicar el número de ronda actual y se inicializa a 0. El cajón a dividir se denota por *Siguiente* y es inicialmente el cajón 0 (el primer cajón). Con N_{Nivel} se denota el número de cajones en el archivo al principio de la ronda $Nivel$. Se puede comprobar fácilmente que $N_{Nivel} = N * 2^{Nivel}$. Sea N el número de cajones al principio de la ronda 0, denotado por N_0 . En la Figura 11.8 se muestra un pequeño archivo asociativo lineal. Cada cajón puede contener cuatro entradas de datos y el archivo contiene inicialmente cuatro cajones, como se muestra en la figura.

**Figura 11.8** Ejemplo de archivo asociativo lineal

Gracias al uso de páginas de desbordamiento se dispone de gran flexibilidad sobre la forma de desencadenar una división. Se puede desencadenar siempre que se añada una página de desbordamiento, o se pueden imponer condiciones adicionales basadas en aspectos tales como la utilización de espacio. Para estos ejemplos se “desencadena” una división cuando al insertar una nueva entrada de datos se crea una página de desbordamiento.

Siempre que se desencadena una división se divide el cajón *Siguiente*, y la función de asociación $h_{Nivel+1}$ redistribuye las entradas entre este cajón (cajón número b) y su imagen dividida; Esta imagen es por tanto el cajón número $b + N_{Nivel}$. Después de dividir un cajón, el valor de *Siguiente* se incrementa en 1. En el archivo de ejemplo, la inserción de la entrada de datos 43* desencadena una división. El archivo después de completar la inserción se muestra en la Figura 11.9.

**Figura 11.9** Después de insertar el registro r con $h(r) = 43$

En cualquier punto en medio de una ronda $Nivel$, todos los cajones anteriores a $Siguiente$ han sido divididos, y el archivo contiene cajones que son sus imágenes divididas, como se ilustra en la Figura 11.7. Los cajones desde $Siguiente$ hasta N_{Nivel} todavía no se han dividido. Si se usa h_{Nivel} sobre una entrada de datos y se obtiene un número b en el rango $Siguiente$ a N_{Nivel} , la entrada pertenece al cajón b . Por ejemplo, $h_0(18)$ es 2 (10 en binario); como este valor está entre los valores actuales de $Siguiente$ (= 1) y N_1 (= 4), este cajón todavía no se ha dividido. Sin embargo, si se obtiene un número b en el rango 0 a $Siguiente$, la entrada de datos puede estar en este cajón o en su imagen dividida (que es el cajón número $b + N_{Nivel}$); debe utilizarse $h_{Nivel+1}$ para determinar a cuál de estos dos cajones pertenece la entrada. En otras palabras, hay que examinar un bit más del valor de asociación de la entrada de datos. Por ejemplo, $h_0(32)$ y $h_0(44)$ son ambos 0 (00 en binario). Como $Siguiente$ actualmente es igual a 1, que indica un cajón que se ha dividido, debe aplicarse h_1 . Se tiene $h_1(32) = 0$ (000 en binario) y $h_1(44) = 4$ (100 en binario). Por lo tanto, 32 pertenece al cajón A y 44 pertenece a su imagen dividida, A2.

Por descontado, no todas las inserciones desencadenan una división. Si se inserta 37^* en el archivo mostrado en la Figura 11.9, el cajón correspondiente tiene espacio para la nueva entrada de datos. El archivo después de la inserción se muestra en la Figura 11.10.

				Nivel=0			
h_1	h_0	PÁGINAS PRIMARIAS				PÁGINAS DE DESBORDAMIENTO	
000	00	32*					
001	01	Siguiente=1	9*	25*	5*	37*	
010	10		14*	18*	10*	30*	
011	11		31*	35*	7*	11*	43*
100	00		44*	36*			

Figura 11.10 Despues de insertar el registro r con $h(r) = 37$

Algunas veces el cajón apuntado por $Siguiente$ (el candidato actual para la división) está lleno, y debe insertarse una nueva entrada de datos en este cajón. En este caso se desencadena una división, por supuesto, pero no se necesita un nuevo cajón de desbordamiento. Esta situación se ilustra insertando 29^* en el archivo mostrado en la Figura 11.10. El resultado es el de la Figura 11.11.

Cuando $Siguiente$ es igual a $N_{Nivel} - 1$ y se desencadena una división, se divide el último de los cajones presentes en el archivo al principio de la ronda $Nivel$. El número de cajones después de la división es el doble que al principio, y se comienza una nueva ronda con $Nivel$ incrementado en 1 y $Siguiente$ inicializado a 0. Incrementar $Nivel$ equivale a duplicar el rango efectivo para el que se asocian las claves. Considérese el archivo de ejemplo de la Figura

		Nivel=0					
h_1	h_0	PÁGINAS PRIMARIAS			PÁGINAS DE DESBORDAMIENTO		
000	00	[32*]					
001	01	[9*]	[25*]				
Siguiente=2		→					
010	10	[14*]	[18*]	[10*]	[30*]		
011	11	[31*]	[35*]	[7*]	[11*]	[43*]	
100	00	[44*]	[36*]				
101	01	[5*]	[37*]	[29*]			

Figura 11.11 Despues de insertar el registro r con $h(r) = 29$

11.12, que se ha obtenido del archivo de la Figura 11.11 insertando 22*, 66* y 34*. (Se anima al lector a intentar desarrollar los detalles de estas inserciones.) Insertar 50* produce una división que lleva a incrementar *Nivel*, como se ha visto anteriormente; el archivo después de esta inserción se muestra en la Figura 11.13.

		Nivel=0					
h_1	h_0	PÁGINAS PRIMARIAS			PÁGINAS DE DESBORDAMIENTO		
000	00	[32*]					
001	01	[9*]	[25*]				
Siguiente=3		→					
010	10	[66*]	[18*]	[10*]	[34*]		
011	11	[31*]	[35*]	[7*]	[11*]	[43*]	
100	00	[44*]	[36*]				
101	01	[5*]	[37*]	[29*]			
110	10	[14*]	[30*]	[22*]			

Figura 11.12 Despues de insertar los registros con $h(r) = 22, 66$ y 34

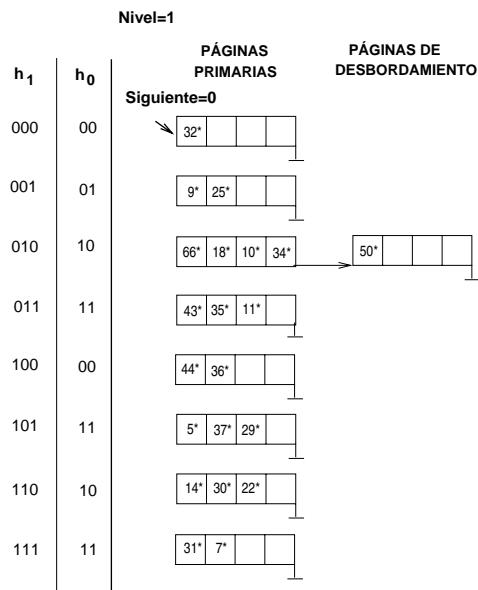


Figura 11.13 Despues de insertar el registro r con $h(r) = 50$

En resumen, una selección por igualdad supone sólo una operación de E/S a no ser que el cajón tenga páginas de desbordamiento; en la práctica, el coste promedio es alrededor de 1,2 accesos a disco para distribuciones razonablemente uniformes. (El coste puede ser considerablemente peor —lineal respecto al número de entradas de datos en el archivo— si la distribución está muy sesgada. La utilización de espacio es también muy pobre con este tipo de distribuciones.) Las inserciones requieren la lectura y escritura de una sola página, excepto cuando se desencadena una división.

No se tratará el borrado en detalle, pero es esencialmente el inverso de la inserción. Si el último cajón del archivo está vacío, puede eliminarse y decrementarse *Siguiente*. (Si *Siguiente* es 0 y el último cajón está vacío, se hace que *Siguiente* apunte al cajón $(M/2) - 1$, donde M es el número actual de cajones, se decremente *Nivel* y se elimina el cajón vacío.) Si se desea, se puede combinar el último cajón con su imagen dividida incluso si no está vacía, utilizando algún criterio para desencadenar esta combinación básicamente de la misma forma. Este criterio normalmente está basado en la ocupación del archivo, y la combinación se puede utilizar para mejorar la utilización del espacio en disco.

11.4 ASOCIACIÓN EXTENSIBLE FRENTE A LINEAL

Para comprender la relación entre la asociación lineal y la extensible se puede imaginar que también se dispone de un directorio en la asociación lineal, con elementos de 0 a $N - 1$. La primera división se produce en el cajón 0, y así se añade el elemento de directorio N . En principio, se puede imaginar que el directorio entero se ha duplicado en este punto; sin embargo, como el elemento 1 es el mismo que el $N+1$, el elemento 2 es el mismo que el $N+2$, y así sucesivamente, se puede evitar la copia real para el resto del directorio. La segunda división

ocurre en el cajón 1; ahora el elemento del directorio $N + 1$ se hace significativo y se añade. Al final de la ronda, todos los N cajones originales se han dividido, y el directorio se ha duplicado en tamaño (porque todos los elementos apuntan a cajones distintos).

Se observa que la elección de las funciones de asociación es realmente muy similar a la asociación extensible —en efecto, pasar de h_i a h_{i+1} en la asociación lineal corresponde a duplicar el directorio en la asociación extensible—. Ambas operaciones duplican el rango efectivo al que se asocian los valores de claves; pero mientras que el directorio se duplica en un único paso de la asociación extensible, pasar de h_i a h_{i+1} en la asociación lineal, junto con el correspondiente duplicado del número de cajones, ocurre gradualmente en el transcurso de una ronda. La idea novedosa detrás de la asociación lineal es que se puede evitar un directorio con una elección inteligente del cajón a dividir. Por otro lado, al dividir siempre el cajón apropiado, la asociación extensible puede llevar a un número de divisiones reducido y una mayor ocupación de los cajones.

La analogía del directorio es útil para comprender las ideas de las asociaciones extensible y lineal. No obstante, la estructura de directorio puede evitarse en la asociación lineal (pero no en la extensible) asignando las páginas primarias de cajones consecutivamente, lo que permitiría localizar la página del cajón i con un simple cálculo de desplazamiento. Para distribuciones uniformes, esta implementación de la asociación lineal tiene un menor coste promedio para las selecciones por igualdad (porque se elimina el nivel de directorio). Para distribuciones sesgadas, esta implementación podría producir cajones vacíos o casi vacíos, cada uno de ellos con al menos una página asignada, dando lugar a un rendimiento pobre respecto a la asociación extensible, que probablemente tendrá una mayor ocupación de cajones.

Una implementación diferente de asociación lineal, en la que se mantiene realmente un directorio, ofrece la flexibilidad de no asignar una página por cajón; se pueden utilizar elementos de directorio *nulos* como en la asociación extensible. Sin embargo, esta implementación introduce el sobrecoste de un nivel de directorio y podría ser costosa para archivos grandes distribuidos uniformemente. (También, aunque esta implementación alivia el problema potencial de una baja ocupación de cajones por no asignar páginas para cajones vacíos, no es una solución completa porque todavía pueden existir muchas páginas con muy pocas entradas.)

11.5 PREGUNTAS DE REPASO

Las respuestas a las preguntas de repaso se pueden encontrar en los apartados indicados.

- ¿Cómo maneja una consulta de igualdad un índice asociativo? Coméntese el uso de la función de asociación para identificar el cajón en el que buscar. Dado un número de cajón, explíquese la forma de localizar el registro en el disco.
- Explíquese la forma en que se gestionan las operaciones de inserción y borrado en un índice asociativo estático. Describáse la forma en que se utilizan las páginas de desbordamiento y su impacto en el rendimiento. ¿Cuántas operaciones de E/S requiere una búsqueda por igualdad, en ausencia de cadenas de desbordamiento? ¿Qué tipos de carga de trabajo gestiona bien un índice asociativo estático y cuándo es especialmente pobre? (Apartado 11.1)

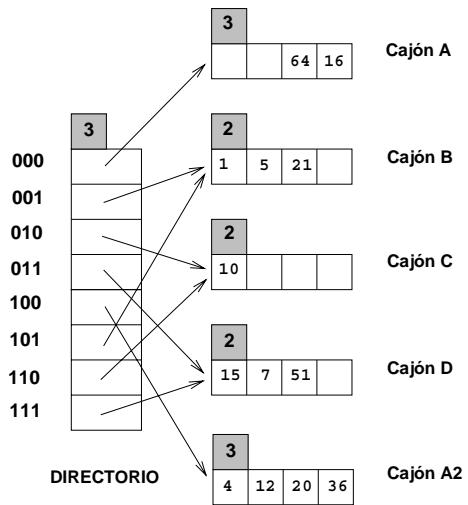


Figura 11.14 Figura para el Ejercicio 11.1

- ¿Cómo utiliza la asociación extensible un directorio de cajones? ¿Cómo gestiona la asociación extensible una consulta por igualdad? ¿Cómo gestiona las operaciones de inserción y borrado? Coméntese en la respuesta la *profundidad global* del índice y la *profundidad local* de un cajón. ¿Bajo qué condiciones puede aumentarse el directorio? (**Apartado 11.2**)
- ¿Qué son las *colisiones*? ¿Por qué son necesarias las páginas de desbordamiento para gestionarlas? (**Apartado 11.2**)
- ¿Cómo la *asociación lineal* evita los directorios? Discútase la división de cajones por turno ratatorio. Explíquese cómo se elige el cajón a dividir y qué desencadena una división. Explíquese el papel de la familia de funciones de asociación, y el de los contadores *Nivel* y *Siguiente*. ¿Cuándo termina una ronda de división? (**Apartado 11.3**)
- Detállese la relación entre la asociación extensible y la lineal. ¿Cuáles son sus méritos relativos? Considérese la utilización de espacio para distribuciones sesgadas, el uso de páginas de desbordamiento para gestionar colisiones en la asociación extensible y el uso de un directorio en la asociación lineal. (**Apartado 11.4**)

EJERCICIOS

Ejercicio 11.1 Considérese el índice asociativo extensible mostrado en la Figura 11.14. Respóndase a las siguientes preguntas sobre este índice:

1. ¿Qué se puede decir sobre la última entrada que fue insertada en el índice?
2. ¿Qué se puede decir sobre la última entrada que fue insertada en el índice si se sabe que no han habido borrados en este índice hasta ahora?
3. Supóngase que se sabe que no se han producido borrados en este índice hasta ahora. ¿Qué se puede decir sobre la última entrada cuya inserción en el índice provocó una división?
4. Muéstrese el índice después de insertar una entrada con valor de asociación 68.

		Nivel=0		PÁGINAS PRIMARIAS	PÁGINAS DE DESBORDAMIENTO
h(1)	h(0)				
000	00			[32 8 24]	
001	01	Siguiente=1	→	[9 25 41 17]	
010	10			[14 18 10 30]	
011	11			[31 35 7 11]	
100	00			[44 36]	

Figura 11.15 Figura para el Ejercicio 11.2

5. Muéstrese el índice después de insertar entradas con valores de asociación 17 y 69 en el árbol original.
6. Muéstrese el índice después de borrar la entrada con valor de asociación 21 en el árbol original. (Supóngase que se utiliza el algoritmo completo de borrado.)
7. Muéstrese el índice después de borrar la entrada con valor de asociación 10 en el árbol original. ¿Se desencadena una combinación de cajones por este borrado? Si no es así, explíquese por qué. (Supóngase que se utiliza el algoritmo completo de borrado.)

Ejercicio 11.2 Considérese el índice asociativo lineal mostrado en la Figura 11.15. Supóngase que se divide siempre que se crea una página de desbordamiento. Respóndase a las siguientes preguntas sobre este índice:

1. ¿Qué se puede decir sobre la última entrada que fue insertada en el índice?
2. ¿Qué se puede decir sobre la última entrada que fue insertada en el índice si se sabe que no han habido borrados en este índice hasta ahora?
3. Supóngase que se sabe que no se han producido borrados en este índice hasta ahora. ¿Qué se puede decir sobre la última entrada cuya inserción en el índice provocó una división?
4. Muéstrese el índice después de insertar una entrada con valor de asociación 4.
5. Muéstrese el índice después de insertar una entrada con valor de asociación 15 en el árbol original.
6. Muéstrese el índice después de borrar las entradas con valores de asociación 36 y 44 en el árbol original. (Supóngase que se utiliza el algoritmo completo de borrado.)
7. Encuéntrese una lista de entradas cuya inserción en el índice original daría lugar a un cajón con dos páginas de desbordamiento. Utilícese el menor número posible de entradas para realizarlo. ¿Cuál es el número máximo de entradas que se pueden insertar en este cajón antes de que ocurra una división que reduzca la longitud de esta cadena de desbordamiento?

Ejercicio 11.3 Respóndase a las siguientes preguntas sobre asociación extensible:

1. Explíquese por qué son necesarias las profundidades local y global.
2. Después de una inserción que hace que el tamaño del directorio se duplique, ¿cuántos cajones tienen exactamente una entrada de directorio apuntándolos? Si después se borra una entrada de uno de esos cajones, ¿qué ocurre con el tamaño de directorio? Explíquense brevemente las respuestas.

3. ¿Garantiza la asociación extensible que se produzca como mucho un acceso a disco para recuperar un registro con un valor de clave dado?
4. Si la función de asociación distribuye las entradas de datos en el espacio de los números de cajón de forma muy sesgada (no uniforme), ¿qué se puede decir del tamaño del directorio? ¿Qué se puede decir de la utilización de espacio en las páginas de datos (es decir, páginas que no son de directorio)?
5. ¿La duplicación requiere que se examinen todos los cajones con profundidad local igual a la profundidad global?
6. ¿Por qué es más difícil gestionar valores de clave duplicados en la asociación extensible que en ISAM?

Ejercicio 11.4 Respóndase a las siguientes preguntas sobre la asociación lineal:

1. ¿Cómo puede una búsqueda promedio tener un coste sólo ligeramente mayor a una operación de E/S en la asociación lineal, dado que los cajones de desbordamiento son parte de su estructura de datos?
2. ¿La asociación lineal garantiza a lo sumo un acceso a disco para recuperar un registro con un valor de clave dado?
3. Si un índice asociativo lineal que utiliza la alternativa (1) para las entradas de datos contiene N registros con P registros por página y una utilización promedio del espacio del 80 por 100, ¿cuál es el coste en el peor caso para una consulta de igualdad? ¿Bajo qué condiciones es éste el coste real de la búsqueda?
4. Si la función de asociación distribuye las entradas de datos sobre el espacio de números de cajón de forma muy sesgada (no uniforme), ¿qué se puede decir sobre la utilización de espacio en las páginas de datos?

Ejercicio 11.5 Proporcióñese un ejemplo de cuándo se puede usar cada elemento (A o B) para cada uno de los siguientes pares “A frente a B”:

1. Un índice asociativo que utilice la alternativa (1) frente a una organización de archivo de montículo.
2. Asociación extensible frente a asociación lineal.
3. Asociación estática frente a asociación lineal.
4. Asociación estática frente a ISAM.
5. Asociación lineal frente a árboles B+.

Ejercicio 11.6 Proporcionéñse ejemplos de lo siguiente:

1. Un índice asociativo lineal y un índice asociativo extensible con las mismas entradas de datos, tales que el índice asociativo lineal tenga más páginas.
2. Un índice asociativo lineal y un índice asociativo extensible con las mismas entradas de datos, tales que el índice asociativo extensible tenga más páginas.

Ejercicio 11.7 Considérese una relación $R(a, b, c, d)$ que contiene un millón de registros, donde cada página de la relación contiene 10 registros. R está organizado como un archivo de montículo con índices no agrupados, y los registros en R están ordenados aleatoriamente. Supóngase que el atributo a es una clave candidata para R, con valores que están en el rango 0 a 999.999. Para cada una de las siguientes consultas, indíquese el enfoque que requeriría más probablemente el menor número de operaciones de E/S para procesarla. Los enfoques a considerar son los siguientes:

- Exploración de todo el archivo de montículo de R.
- Utilización de un índice en árbol B+ sobre el atributo R.a.
- Utilización de un índice asociativo sobre el atributo R.a.

Las consultas son:

1. Determinar todas las tuplas de R.
2. Determinar todas las tuplas de R tales que $a < 50$.

Nivel=0, N=4			
h₁	h₀	PÁGINAS PRIMARIAS	
000	00	Siguiente=0	64 44
001	01		9 25 5
010	10		10
011	11		31 15 7 3

Figura 11.16 Figura del Ejercicio 11.9

3. Determinar todas las tuplas de R tales que $a = 50$.
4. Determinar todas las tuplas de R tales que $a > 50$ y $a < 100$.

Ejercicio 11.8 ¿Cómo cambiarían las respuestas al Ejercicio 11.7 si a no es una clave candidata de R? ¿Cómo cambiarían si se supone que los registros de R están ordenados por a ?

Ejercicio 11.9 Considérese la instantánea del índice asociativo lineal mostrada en la Figura 11.16. Supóngase que ocurre una división de cajón siempre que se crea una página de desbordamiento.

1. ¿Cuál es el número *máximo* de entradas de datos que pueden insertarse (dada la mejor distribución posible para las claves) antes de tener que dividir un cajón? Explíquese muy brevemente.
2. Muéstrese el archivo después de insertar un *único* registro cuya inserción provoca una división de cajón.
3. (a) ¿Cuál es el número *mínimo* de inserciones de registros que producirán una división en los cuatro cajones? Explíquese muy brevemente.
 (b) ¿Cuál es el valor de *Siguiente* después de hacer estas inserciones?
 (c) ¿Qué se puede decir sobre el número de páginas en el cuarto cajón mostrado después de esta serie de inserciones de registros?

Ejercicio 11.10 Considérense las entradas de datos del índice asociativo lineal del Ejercicio 11.9.

1. Muéstrese un índice asociativo extensible con las mismas entradas de datos.
2. Respóndase a las preguntas del Ejercicio 11.9 respecto a este índice.

Ejercicio 11.11 Al contestar a las siguientes preguntas, supóngase que se utiliza el algoritmo de borrado completo. Supóngase que se realiza la combinación cuando un cajón se queda vacío.

1. Proporcióñese un ejemplo de asociación extensible en el que borrar una entrada reduzca la profundidad global.
2. Proporcióñese un ejemplo de asociación lineal en el que borrar una entrada decremente *Siguiente* pero no cambia *Nivel*. Muéstrese el archivo antes y después del borrado.
3. Proporcióñese un ejemplo de asociación lineal en el que borrar una entrada decremente *Nivel*. Muéstrese el archivo antes y después del borrado.

4. Proporcióñese un ejemplo de asociación extensible y una lista de entradas e_1, e_2, e_3 tal que al insertarlas en orden se produzcan tres divisiones y al borrarlas en orden inverso se obtenga el índice original. Si no existe tal ejemplo, explíquese por qué.
5. Proporcióñese un ejemplo de índice asociativo lineal y una lista de entradas e_1, e_2, e_3 tal que al insertarlas en orden se produzcan tres divisiones y al borrarlas en orden inverso se obtenga el índice original. Si no existe tal ejemplo, explíquese por qué.



EJERCICIOS BASADOS EN PROYECTOS

Ejercicio 11.12 (*Nota para los profesores: se deben proporcionar detalles adicionales si se asigna esta pregunta. Véase el Apéndice 30 en el sitio Web del libro.*) Impleméntese asociación lineal o extensible en Minibase.

NOTAS BIBLIOGRÁFICAS

La asociación se trata en detalle en [281]. La asociación extensible se propuso en [166]. Litwin propuso la asociación lineal en [302]. Una generalización de la asociación lineal para entornos distribuidos se describe en [304]. Se ha investigado extensivamente en las técnicas de indexación asociativa. Larson describe dos variaciones de la asociación lineal en [294] y [295]. Ramakrishna presenta un análisis de las técnicas de asociación en [365]. Las funciones de asociación que no producen desbordamientos de cajones se estudian en [366]. Las técnicas de asociación que preservan el orden se describen en [303] y [201]. La asociación particionada, en la que cada campo se asocia para obtener algunos bits de la dirección del cajón, extiende la asociación para el caso de consultas en las que las condiciones de igualdad se especifican solamente para algunos de los campos clave. Este enfoque fue propuesto por Rivest [378] y se trata en [455]; se describe un desarrollo posterior en [369].

PARTE IV

DISEÑO Y AJUSTE DE BASES DE DATOS



12

REFINAMIENTO DE ESQUEMAS Y FORMAS NORMALES

- ¿Qué problemas causa almacenar la información de manera redundante?
- ¿Qué son las dependencias funcionales?
- ¿Qué son las formas normales y cuál es su finalidad?
- ¿Cuáles son las ventajas de la FNBC y de la 3FN?
- ¿Qué se tiene en cuenta al descomponer las relaciones en las formas normales correspondientes?
- ¿Dónde encaja la normalización en el proceso de diseño de las bases de datos?
- ¿Las dependencias más generales resultan útiles para el diseño de bases de datos?
- **Conceptos fundamentales:** redundancia, anomalías de inserción, eliminación y actualización; dependencia funcional, axiomas de Armstrong; cierre de las dependencias, cierre de los atributos; formas normales, FNBC, 3FN; descomposiciones, reunión sin pérdida, conservación de las dependencias; dependencias multivaloradas, dependencias de reunión, dependencias de la inclusión, 4FN, 5FN.

Es una triste verdad que incluso los grandes hombres tienen malas amistades.

— Charles Dickens

El diseño conceptual de bases de datos proporciona un conjunto de esquemas de relaciones y de restricciones de integridad (RI) que se pueden considerar un buen punto de partida para el diseño final de la base de datos. Este diseño inicial se debe refinar mediante la toma en consideración de las RI de una manera más completa de lo que permiten las estructuras del modelo ER y también mediante la toma en consideración de los criterios de rendimiento y de las cargas de trabajo típicas. En este capítulo se estudiará el modo en que las RI se pueden utilizar para refinar el esquema conceptual producido mediante la traducción del diseño del modelo ER en un conjunto de relaciones. La carga de trabajo y las consideraciones de rendimiento se tratan en el Capítulo 13.

Este capítulo se centra en una clase importante de restricciones denominada *dependencias funcionales*. Otras clases de RI, por ejemplo, las *dependencias multivaloradas* y las *dependencias de reunión*, también ofrecen información útil. A veces pueden revelar redundancias que no se pueden detectar sólo mediante las dependencias funcionales. Estas otras restricciones se estudiarán brevemente.

Este capítulo está organizado de la manera siguiente. El Apartado 12.1 es una visión general del enfoque del refinamiento de esquemas que se estudia en este capítulo. En el Apartado 12.2 se introducen las dependencias funcionales. En el Apartado 12.3 se muestra la manera de razonar con la información de dependencia funcional para inferir nuevas dependencias a partir de un conjunto dado de dependencias. En el Apartado 12.4 se introducen las formas normales de las relaciones; la forma normal que satisface cada relación es una medida de la redundancia en esa relación. Las relaciones con redundancia se pueden refinar *descomponiéndolas* o sustituyéndolas por relaciones más pequeñas que contengan la misma información pero sin redundancia. Las descomposiciones y las propiedades que se desea que tengan se estudian en el Apartado 12.5, y se muestra el modo en que se pueden descomponer las relaciones en otras más pequeñas en la forma normal deseable en el Apartado 12.6.

En el Apartado 12.7 se presentan varios ejemplos que ilustran el modo en que los esquemas relacionales obtenidos mediante la traducción de diseños del modelo ER pueden pese a todo sufrir de redundancia, y se analiza la manera de refinar esos esquemas para eliminar el problema. En el Apartado 12.8 se describen otros tipos de dependencias para el diseño de bases de datos. Se concluye con un estudio de la normalización para el caso de estudio, la tienda de Internet, en el Apartado 12.9.

12.1 INTRODUCCIÓN AL REFINAMIENTO DE ESQUEMAS

Ahora se presentará una visión general de los problemas que pretende abordar el refinamiento de esquemas y un enfoque del refinamiento basado en las descomposiciones. El almacenamiento redundante de la información es el origen de estos problemas. Aunque la descomposición puede eliminar la redundancia, puede provocar problemas propios y debe emplearse con cuidado.

12.1.1 Problema causados por la redundancia

Guardar la misma información **de forma redundante**, es decir, en más de un sitio de la base de datos, puede provocar varios problemas:

- **Almacenamiento redundante.** Alguna información se almacena de manera repetida.
- **Anomalías de actualización.** Si se actualiza una copia de esos datos repetidos, se crea una inconsistencia a menos que se actualicen de manera parecida todas las copias.
- **Anomalías de inserción.** Puede que no resulte posible almacenar determinada información a menos que otra información, sin relación alguna con ella, también se almacene.

- **Anomalías de borrado.** Puede que no resulte posible eliminar cierta información sin perder también otra, sin relación alguna con ella.

Considérese una relación obtenida traduciendo una variante del conjunto de entidades Empleados_temp del Capítulo 2:

Empleados_temp(*dni*, *nombre*, *plaza*, *categoría*, *sueldo_hora*, *horas_trabajadas*)

En este capítulo se omite la información del tipo de atributo en aras de la brevedad, ya que nuestra atención se centra en la agrupación de atributos en relaciones. A menudo se abrevia el nombre de un atributo a una sola letra y se hace referencia al esquema de la relación mediante una cadena de letras, una por atributo. Por ejemplo, se hace referencia al esquema Empleados_temp como *DNPCSH* (*S* denota el atributo *sueldo_hora*).

La clave de Empleados_temp es *dni*. Además, supongamos que el atributo *sueldo_hora* viene determinado por el atributo *categoría*. Es decir, para cada valor dado de *categoría* sólo hay un valor admisible de *sueldo_hora*. Esta RI es un ejemplo de *dependencia funcional*. Puede provocar redundancia en la relación Empleados_temp, como puede verse en la Figura 12.1.

<i>dni</i>	<i>nombre</i>	<i>plaza</i>	<i>categoría</i>	<i>sueldo_hora</i>	<i>horas_trabajadas</i>
12322366	Avelino	48	8	10	40
23131536	Sobrino	22	8	10	30
13124365	Somoza	35	5	7	30
43426375	González	35	5	7	32
61267413	Martínez	35	8	10	40

Figura 12.1 Un ejemplar de la relación Empleados_temp

Si aparece el mismo valor en la columna *categoría* de dos tuplas, la RI indica que debe aparecer también el mismo valor en la columna *sueldo_hora*. Esta redundancia tiene las mismas consecuencias negativas que antes:

- *Almacenamiento redundante.* El valor 8 de categoría corresponde al sueldo por hora de 10, y esta asociación se repite tres veces.
- *Anomalías de actualización.* Se podría actualizar el *sueldo_hora* de la primera tupla sin llevar a cabo una modificación similar en la segunda.
- *Anomalías de inserción.* No se puede insertar la tupla de ningún empleado a menos que se sepa el sueldo por hora correspondiente a la categoría de ese empleado.
- *Anomalías de borrado.* Si se eliminan todas las tuplas con una categoría determinada (por ejemplo, las tuplas de Somoza y de González) se pierde la asociación entre el valor de la *categoría* y el del *sueldo_hora*.

Idealmente, se desea disponer de esquemas que no permitan la redundancia pero, como mínimo, se desea poder identificar los esquemas que la permiten. Aunque se elija aceptar un esquema con alguno de estos inconvenientes, quizás debido a consideraciones de rendimiento, es necesario tomar una decisión informada.

Valores nulos

Merece la pena considerar si el empleo de valores nulos (*null*) puede abordar alguno de estos problemas. Como se verá en el contexto del ejemplo, no pueden ofrecer una solución completa, pero aportan algo de ayuda. En este capítulo no se trata el empleo de los valores *null* más allá de este ejemplo.

Considérese el ejemplo de la relación Empleados_temporales. Evidentemente, los valores *null* no pueden ayudar a eliminar el almacenamiento redundante ni las anomalías de actualización. Parece que pueden abordar las anomalías de inserción y de eliminación. Por ejemplo, para tratar con el ejemplo de la anomalía de inserción, se puede insertar una tupla empleado con valores *null* en el campo sueldo por hora. Sin embargo, los valores *null* no pueden abordar todas las anomalías de inserción. Por ejemplo, no se puede registrar el sueldo por hora de ninguna categoría a menos que haya algún empleado con esa categoría, ya que no se pueden almacenar valores nulos en el campo *dni*, que es la clave principal. De manera parecida, para tratar con el ejemplo de anomalía de eliminación, se podría considerar almacenar una tupla con valores *null* en todos los campos salvo *categoría* y *sueldo_hora* si la última tupla con una *categoría* dada se eliminara en caso contrario. Sin embargo, esta solución no funciona porque exige que el valor *dni* sea *null*, y los campos de la clave principal no pueden ser *null*. Por tanto, los valores *null* no ofrecen una solución general para los problemas de redundancia, aunque puedan ayudar en algunos casos.

12.1.2 Descomposiciones

De manera intuitiva, la redundancia surge cuando un esquema relacional fuerza una asociación entre atributos que no es natural. Se pueden utilizar las dependencias funcionales (y, a ese respecto, otras RI) para identificar esas situaciones y sugerir maneras de refinar el esquema. La idea fundamental es que muchos problemas que surgen de la redundancia se pueden abordar sustituyendo una relación dada por un conjunto de relaciones “menores”.

La **descomposición del esquema de una relación *R*** consiste en sustituir el esquema de la relación por dos (o más) esquemas de relación, cada uno de los cuales contiene un subconjunto de los atributos de *R* y, conjuntamente, incluyen todos los atributos de *R*. De manera intuitiva, se desea almacenar la información de cualquier ejemplar dado de *R* almacenando proyecciones de esa ejemplar. Este apartado examina el empleo de las descomposiciones mediante varios ejemplos.

Empleados_temp se puede descomponer en dos relaciones:

Empleados_temp2(dni, nombre, plaza, categoría, horas_trabajadas)
Sueldos(categoría, sueldo_hora)

Los ejemplares de estas relaciones correspondientes al ejemplar de la relación Empleados_temp de la Figura 12.1 puede verse en la Figura 12.2.

Obsérvese que se puede registrar fácilmente el sueldo por hora de cualquier categoría con sólo añadir una tupla a *Sueldos*, aunque no aparezca ningún empleado con esa categoría en el ejemplar actual de Empleados_temp. La modificación del sueldo asociado a una categoría implica actualizar una sola tupla de *Sueldos*. Esto resulta más eficiente que actualizar varias tuplas (como en el diseño original), y elimina la posibilidad de inconsistencias.

<i>dni</i>	<i>nombre</i>	<i>plaza</i>	<i>categoría</i>	<i>horas_trabajadas</i>
123-22-3666	Avelino	48	8	40
231-31-5368	Sobrino	22	8	30
131-24-3650	Somoza	35	5	30
434-26-3751	González	35	5	32
612-67-4134	Martínez	35	8	40

<i>categoría</i>	<i>sueldo hora</i>
8	10
5	7

Figura 12.2 Ejemplares de Empleados_temporales2 y de Sueldos

12.1.3 Problemas relacionados con la descomposición

A menos que se tenga cuidado, la descomposición del esquema de una relación puede crear más problemas de los que soluciona. Se deben formular repetidamente dos preguntas importantes:

1. ¿Hace falta descomponer la relación?
2. ¿Qué problemas (si es que provoca alguno) provoca una descomposición dada?

Para ayudar con la primera pregunta se han propuesto varias *formas normales* para las relaciones. Si el esquema de una relación se halla en alguna de esas formas normales, se sabe que no pueden surgir determinadas clases de problemas. Considerar la forma normal del esquema de una relación dado puede ayudar a decidir si se debe seguir descomponiendo. Si se decide que el esquema de la relación debe seguir descomponiéndose, hay que escoger una descomposición concreta (es decir, un conjunto concreto de relaciones más pequeñas que sustituirá a la relación dada).

Con respecto a la segunda pregunta resultan de especial interés dos propiedades de las descomposiciones. La propiedad de la *reunión sin pérdida* permite recuperar cualquier ejemplar de la relación descompuesta a partir de los ejemplares correspondientes de las relaciones de menor tamaño. La propiedad de la *conservación de las dependencias* permite hacer que se cumpla cualquier restricción de la relación original con sólo hacer que se cumplan algunas restricciones en cada una de las relaciones de menor tamaño. Es decir, no hace falta llevar a cabo reuniones de las relaciones de menor tamaño para comprobar si se viola alguna restricción de la relación original.

Desde el punto de vista del rendimiento, las consultas sobre la relación original pueden exigir que se reúnan las relaciones descompuestas. Si esas consultas son frecuentes, puede que la penalización en rendimiento de la descomposición de la relación no resulte aceptable. En ese caso se puede escoger soportar alguno de los problemas de la redundancia y no descomponer la relación. Es importante tener conciencia de los problemas que puede causar en el diseño esa redundancia residual y tomar medidas para evitarlos (por ejemplo, añadiendo algunas comprobaciones al código de la aplicación). En algunas situaciones la descomposición puede *mejorar* realmente el rendimiento. Esto ocurre, por ejemplo, si la mayor parte de las consultas

y de las actualizaciones sólo examinan una de las relaciones descompuestas, que es menor que la relación original. El efecto de las descomposiciones en el rendimiento de las consultas no se estudia en este capítulo; ese asunto se trata en el Apartado 13.8.

El objetivo de este capítulo es explicar algunos conceptos potentes y diseñar directrices basadas en la teoría de las dependencias funcionales. Un buen diseñador de bases de datos debe tener una clara comprensión de las formas normales y de los problemas que alivian (o no), de la técnica de la descomposición y de los posibles problemas con las descomposiciones. Por ejemplo, el diseñador suele plantear preguntas como las siguientes. ¿Se halla la relación en una forma normal determinada? ¿La descomposición conserva las dependencias? El objetivo es explicar el momento de plantear estas preguntas y el significado de las respuestas.

12.2 DEPENDENCIAS FUNCIONALES

Una **dependencia funcional** (DF) es un tipo de RI que generaliza el concepto de *clave*. Sea R el esquema de una relación y sean X e Y conjuntos no vacíos de atributos de R . Se dice que un ejemplar r de R satisface la DF $X \rightarrow Y$ ¹ si se cumple lo siguiente para todo par de tuplas t_1 y t_2 de r :

Si $t_1.X = t_2.X$, entonces $t_1.Y = t_2.Y$.

Se emplea la notación $t_1.X$ para hacer referencia a la proyección de la tupla t_1 sobre los atributos de X , en una extensión natural de la notación del CRT (véase el Capítulo 4) se usa $t.a$ para hacer referencia al atributo a de la tupla t . La DF $X \rightarrow Y$ indica básicamente que si dos tuplas coinciden en el valor de los atributos de X , también deben coincidir en el valor de los atributos de Y .

La Figura 12.3 ilustra el significado de la DF $AB \rightarrow C$ mostrando un ejemplar que satisface esa dependencia. Las dos primeras tuplas muestran que las DF no son lo mismo que las restricciones de clave: aunque no se viole la DF, es evidente que AB no es clave de la relación. La tercera y la cuarta tuplas ilustran que, si dos tuplas se diferencian en el campo A o en el campo B , pueden diferenciarse en el campo C sin violar la DF. Por otro lado, si se añade la tupla $\langle a1, b1, c2, d1 \rangle$ al ejemplar que aparece en esta figura, el ejemplar resultante viola la DF; para ver esta violación, compárese la primera tupla de la figura con la tupla nueva.

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

Figura 12.3 Un ejemplar que satisface $AB \rightarrow C$

Recuérdese que los ejemplares *legales* de las relaciones deben satisfacer todas las RI especificadas, incluidas todas las DF especificadas. Como se destacó en el Apartado 3.2, las

¹ $X \rightarrow Y$ se lee X determina funcionalmente a Y o, simplemente, X determina a Y .

RI se deben identificar y especificar en términos de la semántica de la empresa real que se esté modelando. Examinando los ejemplares de una relación se puede saber si una DF *no* se cumple. Sin embargo, nunca se puede deducir si una DF *sí* se cumple con sólo mirar uno o varios ejemplares de la relación, ya que las DF, al igual que otras RI, son afirmaciones relativas a *todos* los posibles ejemplares legales de la relación.

Las restricciones de clave principal son un caso especial de DF. Los atributos de la clave desempeñan el papel de X , y el conjunto de todos los atributos de la relación interpretan el de Y . Obsérvese, no obstante, que la definición de DF no exige que el conjunto X sea mínimo; la condición adicional de mínimo debe cumplirse para que X sea clave. Si se cumple que $X \rightarrow Y$, donde Y es el conjunto de todos los atributos y hay algún subconjunto (estrictamente incluido) V de X tal que se cumple que $V \rightarrow Y$, entonces X es *superclave*.

En el resto de este capítulo se verán varios ejemplos de DF que no son restricciones de clave.

12.3 RAZONAMIENTO SOBRE LAS DF

Dado un conjunto de DF sobre el esquema de una relación R , normalmente se cumplen varias DF adicionales sobre R siempre que se cumplan todas las DF dadas. Como ejemplo, considérese:

Trabajadores(*dni*, *nombre*, *plaza*, *idd*, *desde*)

Se sabe que se cumple $dni \rightarrow idd$, ya que dni es la clave, y se da como cierto que se cumple la DF $idd \rightarrow plaza$. Por tanto, en cualquier ejemplar legal de Trabajadores, si dos tuplas tienen el mismo valor de dni , deben tener el mismo valor de idd (debido a la primera DF) y, como tienen el mismo valor de idd , también deben tener el mismo valor de $plaza$ (debido a la segunda DF). Por tanto, también se cumple la DF $dni \rightarrow plaza$ para Trabajadores.

Se dice que la DF d **está implícita en** un conjunto dado D de DF si se cumple d en todos los ejemplares de la relación que satisfacen todas las dependencias de D ; es decir, d se cumple siempre que se cumplen todas las DF de D . Obsérvese que no es suficiente que se cumpla d en algún ejemplar que satisfaga todas las dependencias de D ; por el contrario, d se debe cumplir en *todos los ejemplares* que satisfagan todas las dependencias de D .

12.3.1 Cierre de un conjunto de DF

El conjunto de todas las DF implícitas en un conjunto D dado de DF se denomina **cierre de D**, denotado como F^+ . Una pregunta importante es cómo se puede **inferir**, o calcular, el cierre de un conjunto D dado de DF. La respuesta es sencilla y elegante. Se pueden aplicar reiteradamente las tres reglas siguientes, denominadas **axiomas de Armstrong**, para inferir todas las DF implícitas en un conjunto D de DF. Se utilizan X , Y y Z para denotar los *conjuntos* de atributos sobre el esquema de la relación R :

- **Reflexividad.** Si $X \supseteq Y$, entonces $X \rightarrow Y$.
- **Aumentatividad.** Si $X \rightarrow Y$, entonces $XZ \rightarrow YZ$ para cualquier Z .
- **Transitividad.** Si $X \rightarrow Y$ y $Y \rightarrow Z$, entonces $X \rightarrow Z$.

Teorema 1 Los axiomas de Armstrong son **seguros**, en el sentido de que sólo generan DF de F^+ cuando se aplican al conjunto D de DF. También son **completos**, en el sentido de que la aplicación reiterada de estas reglas genera todas las DF del cierre de F^+ .

La seguridad de los axiomas de Armstrong se prueba sin complicación alguna. La complejidad es más difícil de probar; véase el Ejercicio 12.15.

Resulta conveniente aplicar algunas reglas adicionales para el razonamiento sobre F^+ :

- **Unión.** Si $X \rightarrow Y$ y $X \rightarrow Z$, entonces $X \rightarrow YZ$.
- **Descomposición.** Si $X \rightarrow YZ$, entonces $X \rightarrow Y$ y $X \rightarrow Z$.

Estas reglas adicionales no son esenciales; su seguridad se puede probar mediante los axiomas de Armstrong.

Para ilustrar el empleo de estas reglas de inferencia para las DF, considérese el esquema de la relación ABC con las DF $A \rightarrow B$ y $B \rightarrow C$. En las **DF triviales** el lado derecho sólo contiene atributos que aparecen también en el lado izquierdo; esas dependencias se cumplen siempre debido a la reflexividad. Mediante la reflexividad se pueden generar todas las dependencias triviales, que son de la forma:

$X \rightarrow Y$, donde $Y \subseteq X$, $X \subseteq ABC$ e $Y \subseteq ABC$.

A partir de la transitividad se obtiene que $A \rightarrow C$. A partir de la aumentatividad se obtienen las dependencias no triviales:

$AC \rightarrow BC$, $AB \rightarrow AC$, $AB \rightarrow CB$.

Como ejemplo adicional se utiliza una versión más elaborada de Contratos:

Contratos(idcontrato, idproveedor, id proyecto, iddep, idrepuesto, cantidad, valor)

El esquema de Contratos se denota como $CPYDRNV$. El significado de cada tupla es que el contrato con idcontrato C es un acuerdo por el que el proveedor P (idproveedor) suministrará N ejemplares del repuesto R (idrepuesto) al proyecto Y (id proyecto) asociado con el departamento D (iddep); el valor V de este contrato es igual a valor.

Se sabe que se cumplen las RI siguientes:

1. La id del contrato C es clave: $C \rightarrow CPYDRNV$.
2. Cada proyecto compra un repuesto dado mediante un solo contrato: $YR \rightarrow C$.
3. Cada departamento compra como máximo un repuesto de cada proveedor: $PD \rightarrow R$.

Se cumplen varias DF más en el cierre del conjunto de DF dadas:

De $YR \rightarrow C$, $C \rightarrow CPYDRNV$ y de la transitividad se infiere que $YR \rightarrow CPYDRNV$.

De $PD \rightarrow R$ y la aumentatividad, se infiere que $PDY \rightarrow YR$.

De $PDY \rightarrow YR$, $YR \rightarrow CPYDRNV$ y de la transitividad se infiere que $PDY \rightarrow CPYDRNV$. (Por cierto, aunque pueda parecer tentador hacerlo, *no se puede* concluir que $PD \rightarrow CPYDRNV$, cancelando Y en los dos lados. La inferencia de DF no es como la multiplicación aritmética.)

Se pueden inferir varias DF adicionales que se hallan en el cierre mediante la aumentatividad o la descomposición. Por ejemplo, de $C \rightarrow CPYDRNV$, mediante descomposición, se puede inferir:

$$C \rightarrow C, C \rightarrow P, C \rightarrow Y, C \rightarrow D, \text{etcétera}$$

Finalmente, hay varias DF triviales que se obtienen de la regla de la reflexividad.

12.3.2 Cierre de los atributos

Si sólo se desea comprobar si una dependencia dada, como $X \rightarrow Y$, se halla en el cierre de un conjunto D de DF, se puede lograr de manera eficiente sin necesidad de calcular F^+ . En primer lugar se calcula el **cierre de los atributos** X^+ con respecto a D , que es el conjunto de atributos A tales que $X \rightarrow A$ se puede inferir mediante los axiomas de Armstrong. El algoritmo para el cálculo del cierre de los atributos de un conjunto X de atributos puede verse en la Figura 12.4.

```

cierre =  $X$ ;
repetir hasta que no haya cambios: {
    si hay alguna DF  $U \rightarrow V$  de  $F$  tales que  $U \subseteq \text{cierre}$ ,
        entonces definir  $\text{cierre} = \text{cierre} \cup V$ 
}

```

Figura 12.4 Cálculo del cierre de atributos del conjunto de atributos X

Teorema 2 *El algoritmo de la Figura 12.4 calcula el cierre de los atributos X^+ del conjunto de atributos X con respecto al conjunto de DF D .*

La prueba de este teorema se considera en el Ejercicio 12.13. Este algoritmo se puede modificar para que busque claves comenzando con el conjunto X que contiene un solo atributo y parando en cuanto el *cierre* contenga todos los atributos del esquema de la relación. Variando el atributo inicial y el orden en que el algoritmo considera las DF se pueden obtener todas las claves candidatas.

12.4 FORMAS NORMALES

Dado el esquema de una relación, hay que decidir si se trata de un buen diseño o hace falta descomponerlo en relaciones más pequeñas. Una decisión así debe estar guiada por la comprensión de los problemas, si los hay, que provoca el esquema actual. Para ofrecer esa orientación se han propuesto varias **formas normales**. Si el esquema de una relación se halla en una de estas formas normales, se sabe que no pueden surgir ciertos tipos de problemas.

Las formas normales basadas en las DF son la *primera forma normal (1FN)*, la *segunda forma normal (2FN)*, la *tercera forma normal (3FN)* y la *forma normal de Boyce-Codd (FNBC)*. Estas formas tienen requisitos cada vez más restrictivos: todas las relaciones en

FNBC están también en 3FN, todas las relaciones en 3FN también están en 2FN y todas las relaciones en 2FN están también en 1FN. Una relación se halla en la **primera forma normal** si todos los campos contienen únicamente valores atómicos, es decir, ni listas ni conjuntos. Este requisito se halla implícito en la definición del modelo relacional. Aunque algunos de los sistemas de bases de datos más recientes relajan esta exigencia, en este capítulo se supondrá que se cumple siempre. La 2FN es principalmente de interés histórico. La 3FN y la FNBC son importantes desde el punto de vista del diseño de bases de datos.

Cuando se estudian las formas normales es importante evaluar el papel desempeñado por las DF. Considérese el esquema de la relación R con los atributos ABC . En ausencia de RI cualquier conjunto de tuplas ternarias es un ejemplar legal y no hay posibilidad alguna de redundancia. Por otro lado, supóngase que se tiene la DF $A \rightarrow B$. Ahora bien, si varias tuplas tienen el mismo valor de A , deben tener también el mismo valor de B . Esta posible redundancia se puede predecir empleando la información de la DF. Si se especifican RI más detalladas, puede que también se logre detectar redundancias más sutiles.

Se estudiará sobre todo la redundancia revelada por la información de las DF. En el Apartado 12.8 se tratan unas RI más sofisticadas, denominadas *dependencias multivaloradas* y *dependencias de reunión* y las formas normales basadas en ellas.

12.4.1 La forma normal de Boyce-Codd

Sea R el esquema de una relación, D el conjunto de DF que se cumplen en R , X un subconjunto de los atributos de R y A un atributo de R . R se halla en la **forma normal de Boyce-Codd** si, para cada DF $X \rightarrow A$ de D es cierto una de las afirmaciones siguientes:

- $A \in X$; es decir, es una DF trivial o
- X es una superclave.

De manera intuitiva, en las relaciones en FNBC las únicas dependencias no triviales son aquéllas en las que una clave determina algún atributo. Por tanto, se puede considerar cada tupla como si fuera una entidad o una relación, identificada por una clave y descrita por los demás atributos. Kent (en [270]) expresa esto de manera colorista, si bien sin excesivo rigor: “Cada atributo debe describir [una entidad o una relación identificadas por] la clave, toda la clave y nada más que la clave.” Si se emplean óvalos para denotar atributos o conjuntos de atributos y se dibujan arcos para indicar las DF, las relaciones en FNBC tienen la estructura que puede verse en la Figura 12.5, que considera sólo una clave por sencillez. (Si hay varias claves candidatas, cada una de ellas puede desempeñar el papel de CLAVE en la figura, mientras que los demás candidatos son los que no pertenecen a la clave candidata elegida.)

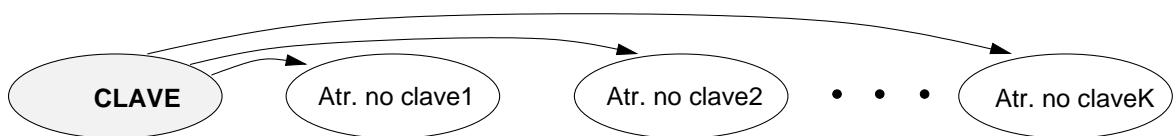


Figura 12.5 DF en una relación en FNBC

La FNBC garantiza que no se pueda detectar redundancia alguna empleando únicamente información sobre las DF. Por tanto, se trata de la forma normal más deseable (desde el punto de vista de la redundancia) si sólo se tiene en cuenta información sobre las DF. Esta consideración se ilustra en la Figura 12.6.

X	Y	A
x	y_1	a
x	y_2	?

Figura 12.6 Ejemplar que ilustra la FNBC

Esta figura muestra (dos tuplas de) un ejemplar de una relación con tres atributos, X , Y y A . Hay dos tuplas con el mismo valor en la columna X . Supóngase ahora que se sabe que este ejemplar satisface la DF $X \rightarrow A$. Es evidente que una de las tuplas tiene el valor a en la columna A . ¿Qué se puede inferir del valor de la columna A de la segunda tupla? Mediante la DF se puede concluir que la segunda tupla también tiene el valor a en esa columna. (Obsérvese que ésta es realmente la única clase de inferencia que se puede hacer sobre el valor de los campos de las tuplas mediante las DF.)

¿Pero esta situación no es un ejemplo de redundancia? Parece que se ha guardado dos veces el valor a . ¿Pueden darse situaciones así en relaciones en FNBC? La respuesta es no. Si esta relación se halla en FNBC, ya que A es diferente de X , se deduce que X debe ser una clave. (En caso contrario, la DF $X \rightarrow A$ violaría la FNBC.) Si X es una clave, entonces $y_1 = y_2$, lo que significa que las dos tuplas son idénticas. Como cada relación, por definición, es un *conjunto* de tuplas, no se puede tener dos copias de la misma tupla y la situación de la Figura 12.6 no puede darse.

Por tanto, si una relación se halla en FNBC, todos los campos de todas las tuplas registran fragmentos de información que no se pueden inferir (empleando sólo DF) del valor de los demás campos de (todas las tuplas del) ejemplar de la relación.

12.4.2 Tercera forma normal

Sea R el esquema de una relación, D el conjunto de DF que se cumplen en R , X un subconjunto de los atributos de R y A un atributo de R . R se halla en la **tercera forma normal** si, para todas las DF $X \rightarrow A$ de D , una de las afirmaciones siguientes es verdadera:

- $A \in X$; es decir, es una DF trivial o
- X es una superclave o
- A es parte de alguna clave de R .

La definición de la 3FN es parecida a la de la FNBC, con la única diferencia de la tercera condición. Todas las relaciones en FNBC se hallan también en 3FN. Para comprender la tercera condición, recuérdese que la clave de una relación es un conjunto *mínimo* de atributos que determina de manera unívoca a todos los demás atributos. A debe formar parte de una clave (cualquier clave, si es que hay varias). No basta con que A forme parte de una superclave,

ya que la última condición la satisfacen todos los atributos. Se sabe que la búsqueda de todas las claves del esquema de una relación es un problema NP completo, igual que el problema de determinar si el esquema de una relación se halla en 3FN.

Supóngase que la dependencia $X \rightarrow A$ causa una violación de la 3FN. Hay dos casos posibles:

- X es subconjunto propio de alguna clave K . Esta dependencia se denomina a veces **dependencia parcial**. En este caso, los pares (X, A) se guardan de manera redundante. Como ejemplo, considérese la relación Reservas con los atributos *MBDT* del Apartado 12.7.4. La única clave es *MBD*, y se tiene la DF $M \rightarrow T$. El número de la tarjeta de crédito de cada marinero se guarda tantas veces como haya reservas para él.
- X no es subconjunto propio de ninguna clave. Estas dependencias se denominan a veces **dependencias transitivas**, ya que indica que se tiene una cadena de dependencias $K \rightarrow X \rightarrow A$. El problema es que no se pueden asociar valores de X con valores de K a menos que se asocien también valores de A con valores de X . Como ejemplo, considérese la relación Empleados_Temp con los atributos *DNPCSH* del Apartado 12.7.1. La única clave es *D*, pero se tiene la DF $C \rightarrow S$, que da lugar a la cadena $D \rightarrow C \rightarrow S$. La consecuencia es que no se puede registrar el hecho de que el empleado *D* tenga la categoría *C* sin conocer el sueldo por hora de esa categoría. Esta condición provoca anomalías de inserción, de eliminación y de actualización.

Las dependencias parciales se ilustran en la Figura 12.7, y las transitivas en la Figura 12.8. Obsérvese que en la Figura 12.8 puede que el conjunto de atributos X no tenga atributos en común con CLAVE; se debe interpretar que el diagrama indica que únicamente que X no es subconjunto de CLAVE.

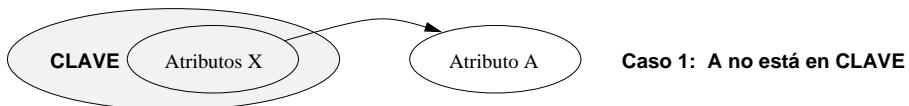


Figura 12.7 Dependencias parciales

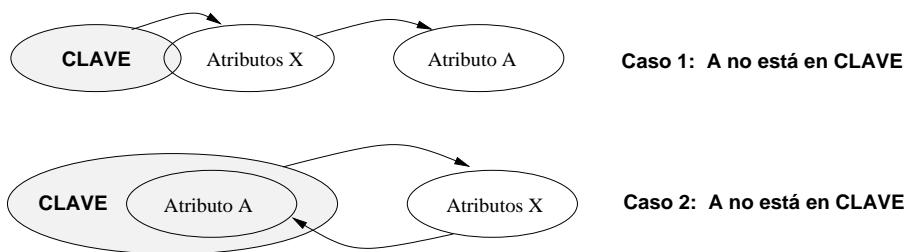


Figura 12.8 Dependencias transitivas

La justificación de la 3FN es más bien técnica. Al hacer una excepción para determinadas dependencias que implican a los atributos de la clave, se puede garantizar que los esquemas

de todas las relaciones se puedan descomponer en conjuntos de relaciones en 3FN empleando sólo descomposiciones que tengan determinadas propiedades deseables (Apartado 12.5). Esa garantía no existe para las relaciones en FNBC; la definición de 3FN relaja las exigencias de la FNBC sólo lo necesario para hacer posible esa garantía. Por tanto, se puede alcanzar un compromiso y aceptar un diseño en 3FN. Como se puede ver en el Capítulo 13, también se puede aceptar a veces ese compromiso (o, incluso, aceptar un esquema que no se halle en 3FN) por otros motivos.

A diferencia de la FNBC, sin embargo, es posible cierta redundancia con la 3FN. Los problemas asociados con las dependencias parciales y con las transitivas persisten si existe alguna dependencia no trivial $X \rightarrow A$ y X no es una superclave, aunque la relación se halle en 3FN porque A sea parte de una clave. Para comprender esto, volvamos a examinar la relación Reservas con los atributos $MBDT$ y la DF $M \rightarrow T$, que afirma que cada marinero utiliza una sola tarjeta de crédito para pagar sus reservas. M no es una clave, y T no forma parte de ninguna clave. (De hecho, la única clave es MBD .) Por tanto, esta relación no se halla en 3FN; los pares (M, T) se almacenan de manera redundante. Sin embargo, si sabemos también que las tarjetas de crédito identifican a su propietario de manera única, se tiene la DF $T \rightarrow M$, que significa que TBD también es clave de Reservas. Por tanto, la dependencia $M \rightarrow T$ no viola la 3FN, y Reservas se halla en 3FN. Pese a todo, en todas las tuplas que contienen el mismo valor de M , se registra de manera redundante el mismo par (M, T) .

Para completar este estudio cabe destacar que la definición de la **segunda forma normal** es básicamente que no se permiten dependencias parciales. Por tanto, si una relación se halla en 3FN (que prohíbe tanto las dependencias parciales como las transitivas), también se encuentra en 2FN.

12.5 PROPIEDADES DE LAS DESCOMPOSICIONES

La descomposición es una herramienta que permite eliminar la redundancia. Como se indicó en el Apartado 12.1.3, sin embargo, es importante comprobar que la descomposición no introduce problemas nuevos. En especial conviene comprobar si la descomposición permite recuperar la relación original y comprobar de modo eficiente las restricciones de integridad. Estas propiedades se estudian a continuación.

12.5.1 Descomposición por reunión sin pérdida

Sea R el esquema de una relación y sea F un conjunto de DF en R . Se dice que una descomposición de R en dos esquemas con los conjuntos de atributos X y Y es una **descomposición por reunión sin pérdida con respecto a F** si, para todos los ejemplares r de R que satisfacen las dependencias de F , $\pi_X(r) \bowtie \pi_Y(r) = r$. En otras palabras, se puede recuperar la relación original a partir de las relaciones descompuestas.

Esta definición se puede ampliar fácilmente para que abarque la descomposición de R en más de dos relaciones. Resulta fácil ver que $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$ se cumple siempre. Sin embargo, lo contrario no se cumple en general. Si se toman las proyecciones de una relación y se recombinan mediante la reunión natural, se suelen obtener tuplas que no se hallaban en la relación original. Esta situación se refleja en la Figura 12.9.

S	P	D
s1	p1	d1
s2	p2	d2
s3	p1	d3

S	P
s1	p1
s2	p2
s3	p1

P	D
p1	d1
p2	d2
p1	d3

El ejemplar r $\pi_{SP}(r)$ $\pi_{PD}(r)$

S	P	D
s1	p1	d1
s2	p2	d2
s3	p1	d3
s1	p1	d3
s3	p1	d1

 $\pi_{SP}(r) \bowtie \pi_{PD}(r)$

Figura 12.9 Ejemplares que ilustran descomposiciones con pérdida

Al sustituir el ejemplar r de la Figura 12.9 por los ejemplares $\pi_{SP}(r)$ y $\pi_{PD}(r)$ se pierde parte de la información. En concreto, supóngase que las tuplas de r denotan relaciones. Ya no se puede decir que las relaciones (s_1, p_1, d_3) y (s_3, p_1, d_1) no se cumplen. La descomposición del esquema SPD en SP y PD , por tanto, tiene pérdida si el ejemplar r que aparece en la figura es legal, es decir, si ese ejemplar puede aparecer en la empresa que se está modelando. (Obsérvense los parecidos entre este ejemplo y el conjunto de relaciones Contratos del 2.5.3.)

Todas las descomposiciones empleadas para eliminar la redundancia deben ser sin pérdida. Esta sencilla comprobación resulta muy útil:

Teorema 3 *Sea R el esquema de una relación y sea D un conjunto de DF que se cumplen en R. La descomposición de R en relaciones con los conjuntos de atributos R_1 y R_2 no tiene pérdida si y sólo si D^+ contiene la DF $R_1 \cap R_2 \rightarrow R_1$ o la DF $R_1 \cap R_2 \rightarrow R_2$.*

En otras palabras, los atributos comunes a R_1 y a R_2 deben contener una clave para R_1 o para R_2 ². Si se descompone una relación en más de dos relaciones se dispone de un algoritmo eficiente (polinómico en el tiempo para el tamaño del conjunto de dependencias) para comprobar si la descomposición tiene pérdida, pero no se estudiará aquí.

Considérese nuevamente la relación Empleados_temp. Tiene los atributos $DNPCSH$ y la DF $C \rightarrow S$ causa una violación de la 3FN. Esta violación se trata descomponiendo la relación en $DNPCSH$ y CS . Como C es común a las dos relaciones descompuestas y se cumple que $C \rightarrow S$, se trata de una descomposición por reunión sin pérdida.

Este ejemplo ilustra una observación general que se deduce del Teorema 3:

Si se cumple la DF $X \rightarrow Y$ en la relación R y $X \cap Y$ es el conjunto vacío, la descomposición de R en $R - Y$ y XY no tiene pérdida.

X aparece tanto en $R - Y$ (ya que $X \cap Y$ es el conjunto vacío) como en XY , y es clave de XY .

Otra observación importante, que se expone sin probarla, tiene que ver con las sucesivas descomposiciones. Supóngase que la relación R se descompone en R_1 y R_2 mediante una descomposición por reunión sin pérdida, y que R_1 se descompone en R_{11} y R_{12} mediante otra descomposición por reunión sin pérdida. Por tanto, la descomposición de R en R_{11} , R_{12}

²Véase el Ejercicio 12.17 para obtener una prueba del Teorema 3. El Ejercicio 12.9 ilustra que la afirmación “sólo si” depende de la suposición de que sólo se puedan especificar como restricciones de integridad las dependencias funcionales.

y $R2$ es una descomposición por reunión sin pérdida; mediante la reunión de $R11$ y $R12$ se puede recuperar $R1$, y reuniendo luego $R1$ y $R2$ se puede recuperar R .

12.5.2 Descomposiciones que conservan las dependencias

Considérese la relación Contratos con los atributos $CPYDRNV$ del Apartado 12.3.1. Las DF dadas son $C \rightarrow CPYDRNV$, $YR \rightarrow C$ y $PD \rightarrow R$. Como PD no es una clave, la dependencia $PD \rightarrow R$ provoca una violación de la FNBC.

Se puede descomponer Contratos en dos relaciones con los esquemas $CPYDRV$ y PDR para evitar esta violación; la descomposición es una descomposición por reunión sin pérdida. Sin embargo, hay un problema sutil. Es fácil hacer que se cumpla la restricción de integridad $YR \rightarrow C$ cuando se inserta una tupla en Contratos asegurándonos de que ninguna tupla ya existente tenga los mismos valores de YR (que la tupla insertada) pero diferentes valores de C . Una vez descompuesta Contratos en $CPYDRV$ y PDR , hacer que se cumpla esta restricción exige una costosa reunión de las dos relaciones siempre que se inserte una tupla en $CPYDRV$. Se dice que esta descomposición no conserva las dependencias.

De manera intuitiva, las *descomposiciones que conservan las dependencias* permiten hacer que se cumplan todas las DF mediante el examen de un solo ejemplar de la relación o de una sola modificación de una tupla. (Téngase en cuenta que las eliminaciones no pueden provocar violaciones de las DF.) Para definir con precisión las descomposiciones que conservan las dependencias hay que introducir el concepto de proyección de las DF.

Sea R el esquema de una relación que se descompone en dos esquemas con los conjuntos de atributos X e Y , y sea D el conjunto de DF de R . La **proyección de D sobre X** es el conjunto de DF del cierre F^+ (no sólo F) que sólo implica atributos de X . La proyección de D sobre los atributos de X se denota como F_X . Obsérvese que la dependencia $U \rightarrow V$ de F^+ sólo se halla en F_X si *todos* los atributos de U y de V se hallan en X .

La descomposición del esquema de la relación R con las DF D en los esquemas con los conjuntos de atributos X e Y **conserva las dependencias** si $(D_X \cup D_Y)^+ = D^+$. Es decir, si se toman las dependencias de D_X y de D_Y y se calcula el cierre de su unión, se vuelven a obtener todas las dependencias del cierre de D . Por tanto, sólo hay que hacer cumplir las dependencias de D_X y de D_Y ; entonces es seguro que se cumplen todas las DF de D^+ . Para hacer que se cumpla D_X sólo hay que examinar la relación X (para las inserciones en esa relación). Para hacer que se cumpla D_Y sólo hay que examinar la relación Y .

Para apreciar la necesidad de considerar el cierre F^+ al calcular la proyección de D , supóngase que la relación R con los atributos ABC se descompone en relaciones con los atributos AB y BC . El conjunto D de DF de R incluye $A \rightarrow B$, $B \rightarrow C$ y $C \rightarrow A$. De ellas, $A \rightarrow B$ se halla en D_{AB} y $B \rightarrow C$ en D_{BC} . ¿Pero conserva esta descomposición las dependencias? ¿Qué ocurre con $C \rightarrow A$? Esta dependencia no se halla implícita en las dependencias mencionadas (hasta ahora) para D_{AB} y D_{BC} .

El cierre de D contiene todas las dependencias de D más $A \rightarrow C$, $B \rightarrow A$ y $C \rightarrow B$. En consecuencia, D_{AB} contiene también $B \rightarrow A$ y D_{BC} contiene $C \rightarrow B$. Por tanto, $D_{AB} \cup D_{BC}$ contiene $A \rightarrow B$, $B \rightarrow C$, $B \rightarrow A$ y $C \rightarrow B$. El cierre de las dependencias de D_{AB} y de D_{BC} incluye ahora $C \rightarrow A$ (que se deduce de $C \rightarrow B$, $B \rightarrow A$ y la transitividad). Así, la descomposición conserva la dependencia $C \rightarrow A$.

Una aplicación directa de la definición da un algoritmo directo para comprobar si una descomposición dada conserva las dependencias. (Este algoritmo es exponencial en el tamaño del conjunto de dependencias. Se dispone de un algoritmo polinómico; véase el Ejercicio 12.7.)

Este apartado comenzó con un ejemplo de descomposición por reunión sin pérdida que no conserva las dependencias. Otras descomposiciones sí conservan las dependencias, pero tienen pérdida. Un ejemplo sencillo consiste en una relación ABC con la DF $A \rightarrow B$ que se descompone en AB y BC .

12.6 NORMALIZACIÓN

Habiendo tratado los conceptos necesarios para comprender el papel de las formas normales y de las descomposiciones en el diseño de bases de datos, ahora se considerarán los algoritmos para transformar las relaciones a la FNBC y a la 3FN. Si el esquema de una relación no se halla en FNBC, es posible obtener una descomposición por reunión sin pérdida en un conjunto de esquemas de relaciones en FNBC. Desafortunadamente, puede que no haya ninguna descomposición que conserve las dependencias en un conjunto de esquemas de relaciones en FNBC. No obstante, siempre hay una descomposición por reunión sin pérdida que conserva las dependencias en un conjunto de esquemas de relaciones en 3FN.

12.6.1 Descomposición en FNBC

A continuación se presentará un algoritmo para descomponer el esquema de una relación R con un conjunto de DF D en un conjunto de esquemas de relación en FNBC:

1. Supóngase que R no se halla en FNBC. Sean $X \subset R$, A un atributo sencillo de R y $X \rightarrow A$ una DF que provoca una violación de la FNBC. Descomponer R en $R - A$ y XA .
2. Si $R - A$ o XA no se hallan en FNBC, descomponer de nuevo mediante la aplicación recursiva de este algoritmo.

$R - A$ denota el conjunto de atributos de R distintos de A y XA denota la unión de los atributos de X y A . Dado que $X \rightarrow A$ viola la FNBC, no se trata de una dependencia trivial; además, A es un único atributo. Por tanto, A no pertenece a X ; es decir, $X \cap A$ es el conjunto vacío. Por tanto, cada descomposición llevada a cabo en el Paso 1 es una descomposición por reunión sin pérdida.

El conjunto de dependencias asociadas con $R - A$ y con XA es la proyección de D sobre sus atributos. Si alguna de las relaciones nuevas no se halla en FNBC, se descompone nuevamente en el Paso 2. Dado que cada descomposición da lugar a relaciones con un número estrictamente menor de atributos, este proceso llega a su fin y deja un conjunto de esquemas de relaciones, todas las cuales se hallan en FNBC. Además, la reunión de los ejemplares de las (dos o más) relaciones obtenidas mediante este algoritmo genera exactamente el ejemplar correspondiente de la relación original (es decir, la descomposición en un conjunto de relaciones que se hallan en FNBC es una descomposición por reunión sin pérdida).

Considérese la relación Contratos con los atributos $CPYDRNV$ y la clave C . Se dan las DF $YR \rightarrow C$ y $PD \rightarrow R$. Empleando la dependencia $PD \rightarrow R$ para guiar la descomposición, se obtienen los dos esquemas PDR y $CPYD NV$. PDR se halla en FNBC. Supóngase que

también se tiene la restricción de que cada proyecto trabaje con un solo proveedor: $Y \rightarrow P$. Esto significa que el esquema $CPYDNNV$ no se halla en FNBC. Por tanto, se vuelve a descomponer en YP y $CYDNNV$. $C \rightarrow YDNNV$ se cumple en $CYDNNV$; las únicas DF que también se cumplen son las obtenidas a partir de esta DF mediante la aumentatividad y, por tanto, todas las DF contienen una clave en su lado izquierdo. Por tanto, cada uno de los esquemas PDR , YP y $CYDNNV$ se hallan en FNBC y este conjunto de esquemas representa también una descomposición por reunión sin pérdida de $CPYDNNV$.

Los pasos de este proceso de descomposición pueden visualizarse como si fueran un árbol, como muestra la Figura 12.10. La raíz es la relación original $CPYDNNV$ y las hojas son las relaciones en FNBC que resultan del algoritmo de descomposición: PDR , YP y $CYDNNV$. De manera intuitiva, cada nodo interno se sustituye por sus hijos mediante un solo paso de descomposición guiado por la DF que se muestra justo debajo de ese nodo.

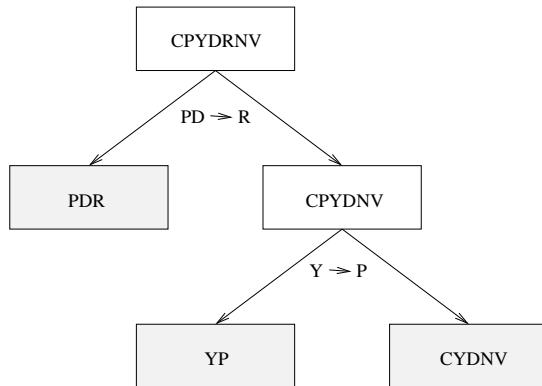


Figura 12.10 Descomposición de $CPYDNNV$ en PDR , YP y $CYDNNV$

Otra perspectiva de la redundancia en FNBC

La descomposición de $CPYDNNV$ en PDR , YP y $CYDNNV$ no conserva las dependencias. De manera intuitiva, no se puede hacer cumplir la dependencia $YR \rightarrow C$ sin una reunión. Una manera de resolver esta situación es añadir una relación con los atributos CYR . En efecto, esta solución equivale a guardar cierta información de manera redundante para hacer que sea menos costoso lograr que se cumpla la dependencia.

Se trata de un asunto util: Cada uno de los esquemas CYR , PDR , YP y $CYDNNV$ se halla en FNBC, y aun así se puede predecir cierta redundancia mediante la información de las DF. En concreto, si se reúnen los ejemplares de la relación para PDR y $CYDNNV$ y se proyecta el resultado sobre los atributos CYR , se debe obtener exactamente el ejemplar guardado en la relación con el esquema CYR . Ya se vio en el Apartado 12.4.1 que no hay tal redundancia dentro de cada relación en FNBC. Este ejemplo muestra que la redundancia puede seguir dándose entre relaciones, aunque no la haya dentro de cada una de ellas.

Alternativas para la descomposición en FNBC

Supóngase que varias dependencias violan la FNBC. Según cuál de esas dependencias se escoja para guiar el siguiente paso de la descomposición, se puede llegar a conjuntos de relaciones en FNBC bastante diferentes. Considérese Contratos. Se acaba de descomponer en *PDR*, *YP* y *CPDNC*. Supóngase que se decide descomponer la relación original *CYDRNV* en *YP* y *CYDRNV*, de acuerdo con la DF $Y \rightarrow P$. Las únicas dependencias que se cumplen en *CYDRNV* son $YR \rightarrow C$ y la dependencia clave $C \rightarrow CYDRNV$. Dado que *YR* es una clave, *CYDRNV* se halla en FNBC. Por tanto, los esquemas *YP* y *CYDRNV* representan una descomposición por reunión sin pérdida de Contratos en relaciones que se hallan en FNBC.

La lección que se puede extraer de esto es que la teoría de las dependencias puede indicar cuándo hay redundancia y da pistas sobre las descomposiciones posibles para abordar el problema, pero no puede discriminar entre las alternativas de descomposición. El diseñador tiene que considerar las diferentes alternativas y escoger una de acuerdo con la semántica de la aplicación.

La FNBC y la conservación de las dependencias

A veces, simplemente no hay ninguna descomposición en FNBC que conserve las dependencias. Por ejemplo, considérese el esquema de la relación *MBD*, en la que cada tupla denota que el marinero *M* ha reservado el barco *B* el día *D*. Si se tienen las DF $MB \rightarrow D$ (cada marinero puede reservar un barco dado, como máximo, durante un día) y $D \rightarrow B$ (cualquier día dado se puede reservar, como máximo, un barco), *MBD* no se halla en FNBC, ya que *D* no es una clave. Si se intenta descomponer, no obstante, no se puede conservar la dependencia $MB \rightarrow D$.

12.6.2 Descomposición en 3FN

Evidentemente, el enfoque que se ha esbozado para la descomposición por reunión sin pérdida en FNBC también da una descomposición por reunión sin pérdida en 3FN. (Generalmente se puede parar un poco antes si se está satisfecho con un conjunto de relaciones en 3FN.) Pero este enfoque no garantiza la conservación de las dependencias.

Una sencilla modificación, sin embargo, ofrece una descomposición por reunión sin pérdida en relaciones en 3FN y que conserva las dependencias. Antes de describir esta modificación hay que introducir el concepto de recubrimiento mínimo de un conjunto de DF.

Recubrimiento mínimo de un conjunto de DF

El **recubrimiento mínimo** de un conjunto *D* de DF es el conjunto *G* de DF tales que:

1. Todas las dependencias de *G* son de la forma $X \rightarrow A$, donde *A* es un atributo sencillo.
2. El cierre D^+ es igual al cierre G^+ .

3. Si se obtiene un conjunto de dependencias H a partir de G mediante la eliminación de una o más dependencias o mediante la eliminación de atributos de alguna dependencia de G , entonces $D^+ \neq H^+$.

De manera intuitiva, el recubrimiento mínimo del conjunto D de DF es el conjunto de dependencias equivalente que es *mínimo* en dos sentidos. (1) Todas las dependencias son tan pequeñas como es posible; es decir, cada atributo del lado izquierdo es necesario y el lado derecho es un solo atributo. (2) Todas las dependencias del recubrimiento mínimo son necesarias para que el cierre sea igual a D^+ .

Por ejemplo, sea D el conjunto de dependencias:

$$A \rightarrow B, ABCD \rightarrow E, EF \rightarrow G, EF \rightarrow H \text{ y } ACDF \rightarrow EG.$$

En primer lugar se reescribe $ACDF \rightarrow EG$ de modo que el lado derecho sea un solo atributo:

$$ACDF \rightarrow E \text{ y } ACDF \rightarrow G.$$

A continuación se considera $ACDF \rightarrow G$. Esta dependencia está implícita en las siguientes DF:

$$A \rightarrow B, ABCD \rightarrow E \text{ y } EF \rightarrow G.$$

Por tanto, se puede eliminar. De manera parecida, se puede eliminar $ACDF \rightarrow E$. Considerese ahora $ABCD \rightarrow E$. Dado que se cumple que $A \rightarrow B$, se puede sustituir por $ACD \rightarrow E$. (En este momento el lector debe comprobar que todas las DF restantes son mínimas y necesarias.) Por tanto, un recubrimiento mínimo de D es el conjunto:

$$A \rightarrow B, ACD \rightarrow E, EF \rightarrow G \text{ y } EF \rightarrow H.$$

El ejemplo anterior ilustra un algoritmo general para la obtención de recubrimientos mínimos de los conjuntos D de DF:

1. **Poner las DF en una forma normalizada.** Obtener un conjunto G de DF equivalentes con un solo atributo en el lado derecho (mediante el axioma de descomposición).
2. **Minimizar el lado izquierdo de cada DF.** Para cada DF de G , comprobar cada atributo del lado izquierdo para ver si se puede eliminar sin dejar de conservar la equivalencia con D^+ .
3. **Eliminar las DF redundantes.** Comprobar cada DF restante de G para ver si se puede eliminar sin dejar de conservar la equivalencia con D^+ .

Obsérvese que el orden en que se consideran las DF al aplicar estos pasos puede generar diferentes recubrimientos mínimos; puede haber varios recubrimientos mínimos diferentes para cada conjunto dado de DF.

Lo que es más importante, es necesario minimizar el lado izquierdo de las DF *antes* de comprobar si hay DF redundantes. Si se invierte el orden de estos dos pasos el conjunto final de DF puede seguir conteniendo alguna DF redundante (es decir, no será un recubrimiento mínimo), como ilustra el ejemplo siguiente. Sea D el conjunto de dependencias, cada una de las cuales se halla en la forma normalizada:

$$ABCD \rightarrow E, E \rightarrow D, A \rightarrow B \text{ y } AC \rightarrow D.$$

Obsérvese que ninguna de estas DF es redundante; si se buscaran en primer lugar DF redundantes se obtendría el mismo conjunto de DF F . El lado izquierdo de $ABCD \rightarrow E$ se puede sustituir por AC sin dejar de conservar la equivalencia con D^+ , y aquí se detendría el proceso si se buscaran DF redundantes en D antes de minimizar el lado izquierdo de las dependencias. Sin embargo, el conjunto de DF que se tiene no es un recubrimiento mínimo:

$$AC \rightarrow E, E \rightarrow D, A \rightarrow B \text{ y } AC \rightarrow D.$$

De acuerdo con la transitividad, las dos primeras DF implican la última que, por tanto, se puede eliminar sin dejar de conservar la equivalencia con D^+ . Lo destacable es que $AC \rightarrow D$ sólo pasa a ser redundante después de que se sustituya $ABCD \rightarrow E$ por $AC \rightarrow E$. Si se minimiza el lado izquierdo de las DF en primer lugar y luego se buscan DF redundantes, se escogen las tres primeras DF de la lista anterior, que constituyen realmente un recubrimiento mínimo para D .

Descomposición en 3FN y conservación de dependencias

Volviendo al problema de obtener una descomposición por reunión sin pérdida en relaciones que se hallen en 3FN que conserve las dependencias, sea R una relación con un conjunto D de DF que es recubrimiento mínimo y sea R_1, R_2, \dots, R_n una descomposición por reunión sin pérdida de R . Para $1 \leq i \leq n$, supóngase que cada R_i se halla en 3FN y que D_i denota la proyección de D sobre los atributos de R_i . Se debe hacer lo siguiente:

- Identificar el conjunto N de las dependencias de D que no se **conservan**, es decir, que no está incluido en el cierre de la unión de D_i s.
- Para cada DF $X \rightarrow A$ de N , crear el esquema de una relación XA y añadirlo a la descomposición de R .

Evidentemente, todas las dependencias de D se conservan si se sustituye R por las R_i s más los esquemas de la forma XA añadidos en este paso. Se deduce que las R_i s se hallan en 3FN. Se puede probar que cada uno de los esquemas XA se halla en 3FN de la manera siguiente. Dado que $X \rightarrow A$ pertenece al recubrimiento mínimo D , $Y \rightarrow A$ no se cumple para ninguna Y que sea un subconjunto estricto de X . Por tanto, X es una clave de XA . Además, si se cumple alguna otra dependencia en XA , el lado derecho sólo puede implicar atributos de X , ya que A es un único atributo (debido a que $X \rightarrow A$ es una DF de un recubrimiento mínimo). Dado que X es una clave de XA , ninguna de esas dependencias adicionales provoca violaciones de la 3FN (aunque puede que sí provoquen alguna violación de la FNBC).

Como optimización, si el conjunto N contiene varias DF con el mismo lado izquierdo, digamos, $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n$, se pueden sustituir por una sola DF equivalente, $X \rightarrow A_1 \dots A_n$. Por tanto, se genera un solo esquema de relación $XA_1 \dots A_n$, en lugar de varios esquemas XA_1, \dots, XA_n , lo que suele resultar preferible.

Considérese la relación Contratos con los atributos $CPYDRNV$ y las DF $YR \rightarrow C, PD \rightarrow R$ e $Y \rightarrow P$. Si se descompone $CPYDRNV$ en PDR y $CPYDNP$, entonces PDR se halla en FNBC, pero $CPYDNP$ no se halla ni siquiera en 3FN. Por tanto, se descompone nuevamente

en YP y $CYDNY$. Los esquemas de relación PDR , YP y $CYDNY$ se hallan en 3FN (de hecho, se hallan en FNBC) y se trata de una descomposición por reunión sin pérdida. Sin embargo, no se conserva la dependencia $YR \rightarrow C$. Este problema se puede abordar añadiendo el esquema de relación CYR a la descomposición.

Síntesis de la 3FN

Se ha dado por supuesto que el proceso de diseño comienza con un diagrama ER, y que el empleo de las DF es, sobre todo, para orientar las decisiones sobre la descomposición. El algoritmo para la obtención de descomposiciones por reunión sin pérdida que conserven las dependencias se ha presentado en el apartado anterior desde este punto de vista —la descomposición por reunión sin pérdida es directa y el algoritmo aborda la conservación de las dependencias añadiendo más esquemas de relaciones—.

Un enfoque alternativo, denominado **síntesis**, es tomar todos los atributos de la relación original R y el recubrimiento mínimo D de las DF que se cumplen en ella y añadir el esquema de la relación XA a la descomposición de R para cada DF $X \rightarrow A$ de D .

El conjunto resultante de esquemas de relaciones se halla en 3FN y conserva todas las DF. Si no se trata de una descomposición por reunión sin pérdida de R , se puede lograr que lo sea añadiendo el esquema de una relación que contenga sólo los atributos que aparecen en alguna clave. Este algoritmo da una descomposición por reunión sin pérdida que conserva las dependencias en 3FN y que tiene complejidad polinómica —se dispone de algoritmos polinómicos para el cálculo de recubrimientos mínimos, y se puede hallar una clave en un tiempo polinómico (aunque se sepaa que hallar todas las claves sea NP completo)—. La existencia de algoritmos polinómicos para la obtención de descomposiciones por reunión sin pérdida que conservan las dependencias en 3FN resulta sorprendente cuando se considera que probar si un esquema dado se halla en 3FN es NP completo. Por ejemplo, considérese la relación ABC con las DF $D = \{A \rightarrow B, C \rightarrow B\}$. El primer paso proporciona los esquemas de relación AB y BC . No se trata de una descomposición por reunión sin pérdida de ABC ; $AB \cap BC$ es B , y ni $B \rightarrow A$ ni $B \rightarrow C$ pertenecen a D^+ . Si se añade el esquema AC , se obtiene también la propiedad de reunión sin pérdida. Aunque el conjunto de relaciones AB , BC y AC es una descomposición por reunión sin pérdida que conserva las dependencias de ABC , se ha obtenido mediante un proceso de **síntesis**, en vez de mediante un proceso de descomposición reiterada. Hay que destacar que la descomposición producida por el enfoque de síntesis depende mucho del recubrimiento mínimo empleado.

Como ejemplo adicional del enfoque de síntesis, considérese la relación Contratos con los atributos $CPYDRNV$ y las DF siguientes:

$$C \rightarrow CPYDRNV, YR \rightarrow C, PD \rightarrow R \text{ e } Y \rightarrow P.$$

Este conjunto de DF no constituye un recubrimiento mínimo y, por tanto, hay que buscar uno. En primer lugar, se sustituye $C \rightarrow CPYDRNV$ por las DF:

$$C \rightarrow P, C \rightarrow Y, C \rightarrow D, C \rightarrow R, C \rightarrow N \text{ y } C \rightarrow V.$$

La DF $C \rightarrow R$ está implícita en $C \rightarrow P$, $C \rightarrow D$ y $PD \rightarrow R$; por lo que se puede eliminar. La DF $C \rightarrow P$ está implícita en $C \rightarrow Y$ e $Y \rightarrow P$; por lo que se puede eliminar. Esto resulta en el recubrimiento mínimo:

$$C \rightarrow Y, C \rightarrow D, C \rightarrow N, C \rightarrow V, YR \rightarrow C, PD \rightarrow R \text{ e } Y \rightarrow P.$$

Mediante el algoritmo para garantizar la conservación de la dependencia se obtiene el esquema relacional CY, CD, CN, CV, CYR, PDR e YP . Este esquema se puede mejorar combinando en $CDYRNV$ las relaciones para las que C es la clave. Además, se tiene en la descomposición PDR e YP . Dado que una de estas relaciones ($CDYRNV$) es una superclave, se ha acabado.

Comparando esta descomposición con la obtenida anteriormente en este apartado, puede verse que son muy parecidas, con la única diferencia de que una de ellas tiene $CDYRNV$ en lugar de CYR y $CYDNV$. En general, sin embargo, puede haber diferencias significativas.

12.7 REFINAMIENTO DE ESQUEMAS EN EL DISEÑO DE BASES DE DATOS

Se ha visto la manera en que la normalización puede eliminar la redundancia y se han analizado varios enfoques de la normalización de las relaciones. Ahora se considerará el modo en que se aplican estas ideas en la práctica.

Los diseñadores de bases de datos suelen emplear una metodología de diseño conceptual, como el diseño ER, para llegar a un primer diseño de las bases de datos. Dado esto, es probable que el enfoque de las descomposiciones reiteradas para rectificar casos de redundancia sea el empleo más natural de las DF y de las técnicas de normalización.

En este apartado se justifica la necesidad de que una etapa de depuración de los esquemas siga al diseño ER. Resulta natural preguntarse si hace falta descomponer las relaciones producidas mediante la traducción de los diagramas ER. ¿No debería llevar un buen diseño ER a un conjunto de relaciones libres de problemas de redundancia? Desafortunadamente, el diseño ER es un proceso subjetivo y complejo, y determinadas restricciones no se pueden expresar en forma de diagramas ER. Se pretende que los ejemplos de este apartado ilustren la razón de que se pueda necesitar la descomposición de las relaciones generadas mediante el diseño ER.

12.7.1 Restricciones en los conjuntos de entidades

Considérese nuevamente la relación `Empleados_temp`. La restricción de que el atributo `dni` es una clave se puede expresar en forma de DF:

$$\{ \text{dni} \} \rightarrow \{ \text{dni}, \text{nombre}, \text{plaza}, \text{categoría}, \text{sueldo_hora}, \text{horas_trabajadas} \}$$

Por abreviar, esta DF se escribe como $D \rightarrow DNPCSH$, empleando una sola letra para denotar cada atributo y omitiendo las llaves, pero el lector debe recordar que los dos lados de las DF contienen conjuntos de atributos. Además, la restricción de que el atributo `sueldo_hora` está determinado por el atributo `categoría` es una DF: $C \rightarrow S$.

Como se vio en el Apartado 12.1.1, esta DF provocaba el almacenamiento redundante de asociaciones categoría-sueldo. *No se puede expresar en términos del modelo ER. Sólo se pueden expresar en el modelo ER las DF que determinan todos los atributos de la relación (por ejemplo, las restricciones de clave).* Por tanto, no se podía detectar cuando se consideró `Empleados_temp` como conjunto de entidades durante el modelado ER.

Se podría aducir que el problema del diseño original era una estructura o un mal diseño ER, que se podría haber evitado introduciendo el conjunto de entidades denominado Tabla_salarial (con los atributos *categoría* y *sueldo hora*) y el conjunto de relaciones Tiene_sueldo que asocia Empleados_temp con Tabla_salarial. Lo importante, sin embargo, es que se podría llegar fácilmente al diseño original dada la naturaleza subjetiva del modelado ER. Resulta muy útil tener técnicas formales para identificar el problema de este diseño y orientarnos hacia un diseño mejor. El valor de esas técnicas no se puede subestimar al diseñar esquemas de gran tamaño —no son infrecuentes los esquemas con más de cien tablas—.

12.7.2 Restricciones en los conjuntos de relaciones

El ejemplo anterior ilustraba la manera en que las DF pueden ayudar a depurar las decisiones subjetivas tomadas durante el diseño ER, pero se podría aducir que el mejor diagrama ER posible habría conducido al mismo conjunto final de relaciones. El ejemplo siguiente muestra el modo en que la información de las DF puede llevar a un conjunto de relaciones al que resulta improbable llegar únicamente mediante el diseño ER.

Recuérdese el ejemplo del Capítulo 2. Supóngase que se dispone de los conjuntos de entidades Repuestos, Proveedores y Departamentos, así como el conjunto de relaciones Contratos, que los implica a todos. Se hace referencia al esquema de Contratos como *CNRPD*. El contrato con identificador *C* especifica que el proveedor *P* proporcionará la cantidad *N* del repuesto *R* al departamento *D*. (Se ha añadido el campo identificador de contrato *C* a la versión de la relación Contratos estudiada en el Capítulo 2.)

Se podría tener la política de que cada departamento compre, como máximo, un repuesto de cada proveedor. Por tanto, si hay varios contratos entre un proveedor y un departamento, se sabe que debe estar implicado en todos ellos el mismo repuesto. Esta restricción es la DF $DP \rightarrow R$.

Una vez más se tiene redundancia y los problemas asociados a ella. Esta situación se puede abordar descomponiendo Contratos en dos relaciones con los atributos *CBPD* y *PDR*. De manera intuitiva, la relación *PDR* registra el repuesto suministrado al departamento en cuestión por un proveedor, y la relación *CNPD* registra información adicional sobre el contrato. Resulta improbable que se alcance un diseño así solamente mediante el modelado ER, ya que es difícil formular entidades o relaciones que se correspondan de modo natural con *CNPD*.

12.7.3 Identificación de los atributos de las entidades

Este ejemplo ilustra el modo en que un examen detenido de las DF puede llevar a una mejor comprensión de las entidades y las relaciones subyacentes a las tablas relacionales; en concreto, muestra que los atributos se pueden asociar fácilmente con el conjunto de entidades “equivocado” durante el diseño ER. El diagrama ER de la Figura 12.11 muestra el conjunto de relaciones denominado Trabaja_en, que es parecido al conjunto de relaciones Trabaja_en del Capítulo 2, pero con la restricción de clave adicional que indica que cada empleado puede trabajar, como máximo, en un departamento. (Obsérvese la flecha que conecta Empleados y Trabaja_en.)

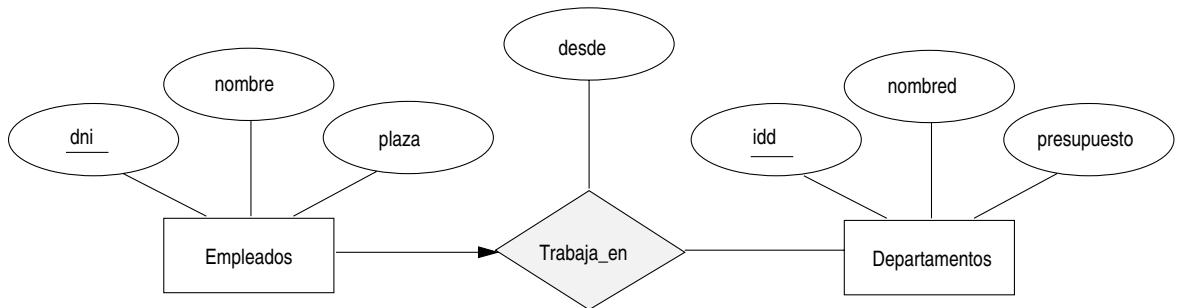


Figura 12.11 El conjunto de relaciones Trabaja_en

Mediante la restricción de la clave se puede traducir este diagrama ER en dos relaciones:

Trabajadores(dni, nombre, plaza, idd, desde)
 Departamentos(idd, nombrd, presupuesto)

El conjunto de entidades Empleados y el conjunto de relaciones Trabaja_en se asignan a una sola relación, Trabajadores. Esta traducción se base en el segundo enfoque estudiado en el Apartado 2.4.1.

Supóngase ahora que se asignan plazas de aparcamiento a los empleados de acuerdo con el departamento en el que trabajan, y que todos los empleados de un departamento dado son asignados a la misma plaza. Esta restricción no es expresable con respecto al diagrama ER de la Figura 12.11. Se trata de otro ejemplo de DF: $idd \rightarrow plaza$. La redundancia de este diseño se puede eliminar descomponiendo la relación Trabajadores en dos:

Trabajadores2(dni, nombre, idd, desde)
 Plazas_dep(idd, plaza)

El nuevo diseño tiene muchas cosas que lo hacen recomendable. Se pueden cambiar las plazas asociadas con cada departamento actualizando una sola tupla de la segunda relación (es decir, no hay anomalías de actualización). Se puede asociar una plaza con un departamento aunque en ese momento no tenga ningún empleado, sin necesidad de utilizar valores *null* (es decir, no hay anomalías de eliminación). Se puede añadir un empleado a un departamento insertando una tupla en la primera relación, aunque no haya ninguna plaza asociada con ese departamento (es decir, no hay anomalías de inserción).

Examinando las dos relaciones, Departamentos y Plazas_dep, que tienen la misma clave, resulta evidente que una tupla de Departamentos y otra de Plazas_dep con el mismo valor de la clave describen las mismas entidades. Esta observación se refleja en el diagrama ER que puede verse en la Figura 12.12.

La traducción de este diagrama al modelo relacional daría como resultado:

Trabajadores2(dni, nombre, idd, desde)
 Departamentos(idd, nombrd, presupuesto, plaza)

Parece intuitivo asociar las plazas con los empleados; por otro lado, las RI revelan que, en este ejemplo, las plazas se hallan asociadas en realidad con los departamentos. El proceso

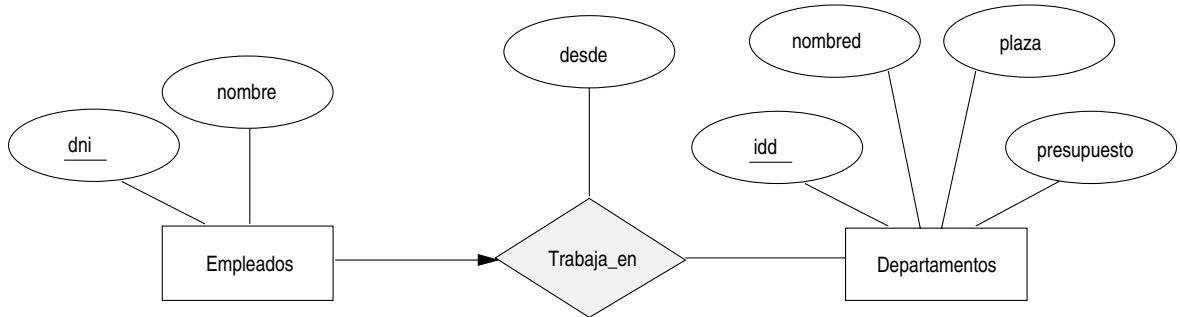


Figura 12.12 El conjunto de relaciones *Trabaja_en* refinado

subjetivo del modelado ER puede pasar esto por alto. El proceso riguroso de la normalización, no.

12.7.4 Identificación de los conjuntos de entidades

Considérese una variante del esquema Reservas empleado en capítulos anteriores. Supóngase que Reservas contiene los atributos *M*, *B* y *D* igual que antes, que indican que el marinero *M* tiene una reserva para el barco *B* el día *D*. Además, supóngase que hay un atributo *T* que denota la tarjeta de crédito en la que se carga la reserva. Este ejemplo se emplea para ilustrar el modo en que se puede emplear la información de las DF para refinar los diseños ER. En concreto, se examinará el modo en que la información de las DF puede ayudar a decidir si los conceptos se deben modelar como entidades o como atributos.

Supóngase que cada marinero emplea una sola tarjeta de crédito para las reservas. Esta restricción se expresa mediante la DF $M \rightarrow T$. Esta restricción indica que, en la relación Reservas, se guarda el número de la tarjeta de crédito de cada marinero tan a menudo como haya reservas de ese marinero, y se tienen redundancia y posibles anomalías de actualización. Una solución es descomponer Reservas en dos relaciones con los atributos *MBD* y *MT*. De manera intuitiva, una guarda la información sobre las reservas y la otra guarda la información sobre las tarjetas de crédito.

Resulta instructivo meditar acerca del diseño ER que llevaría a esas relaciones. Un enfoque es introducir el conjunto de entidades denominado Tarjetas_crédito, con el atributo único *númtarjeta*, y el conjunto de relaciones Tiene_tarjeta que asocia Marineros y Tarjetas_crédito. Al recordar que cada tarjeta de crédito pertenece a un solo marinero se pueden asignar Tiene_tarjeta y Tarjetas_crédito a una sola relación con los atributos *MT*. Probablemente los números de las tarjetas de crédito no se modelarían como entidades si el principal motivo de nuestro interés en ellos fuera indicar el modo en que se va a pagar cada reserva; en ese caso bastaría con emplear un atributo para modelarlos.

Un segundo enfoque es hacer de *númtarjeta* atributo de Marineros. Pero este enfoque no resulta muy natural —puede que cada marinero tenga varias tarjetas, y no interesan todas—. Interesa la tarjeta que se emplea para pagar las reservas, que se modela mejor como atributo de la relación Reservas.

Una manera útil de considerar el problema de diseño de este ejemplo es hacer en primer lugar de *númtarjeta* atributo de Reservas y refinar luego las tablas resultantes teniendo en cuenta la información sobre las DF. (Que el diseño se refine añadiendo a la tabla obtenida de Marineros *númtarjeta* o creando una tabla nueva con los atributos *MT* es otro asunto diferente.)

12.8 OTROS TIPOS DE DEPENDENCIAS

Puede que las DF sean el tipo de restricción más frecuente e importante desde el punto de vista del diseño de bases de datos. Sin embargo, hay varios tipos más de dependencias. En concreto, hay una teoría bien desarrollada del diseño de bases de datos mediante las *dependencias multivaloradas* y las *dependencias de reunión*. Al tener en cuenta esas dependencias se pueden identificar posibles problemas de redundancia que no se pueden detectar empleando sólo DF.

Este apartado ilustra los tipos de redundancia que se pueden detectar mediante las dependencias multivaloradas. La principal observación, no obstante, es que unas directrices sencillas (que se pueden seguir empleando tan sólo razonamientos de DF) pueden indicar si hace falta considerar restricciones complejas como las multivaloradas y las dependencias de reunión. También se comenta el papel de las *dependencias de inclusión* en el diseño de bases de datos.

12.8.1 Dependencias multivaloradas

Supóngase que se tiene una relación con los atributos *asignatura*, *profesor* y *libro*, que denotamos como *APL*. El significado de las tuplas es que el profesor *P* puede impartir la asignatura *A*, y que el libro *L* es un texto recomendado para esa asignatura. No hay DF; la clave es *APL*. Sin embargo, los textos recomendados para cada asignatura son independientes del profesor. El ejemplo de la Figura 12.13 ilustra esta situación.

<i>asignatura</i>	<i>profesor</i>	<i>libro</i>
Física101	Gómez	Mecánica
Física101	Gómez	Óptica
Física101	Bruno	Mecánica
Física101	Bruno	Óptica
Matemáticas301	Gómez	Mecánica
Matemáticas301	Gómez	Vectores
Matemáticas301	Gómez	Geometría

Figura 12.13 Relación en FNBC con redundancia que revelan las DM

Cabe destacar tres aspectos:

- El esquema de la relación *APL* se halla en FNBC; por tanto, no se pensaría en descomponerla más si sólo se tuviesen en cuenta las DF que se cumplen en *APL*.

- Hay redundancia. El hecho de que Gómez pueda enseñar Física101 se registra una vez por cada texto recomendado para esa asignatura. De manera análoga, el hecho de que Óptica sea uno de los textos de Física101 se registra una vez por cada posible profesor.
- La redundancia se puede eliminar descomponiendo APL en AP y en AL .

La redundancia de este ejemplo se debe a la restricción de que los textos de las asignaturas sean independientes de los profesores, lo cual no se puede expresar en términos de DF. Esta restricción es un ejemplo de *dependencia multivalorada* o DM. En teoría esta situación se debería modelar empleando dos conjuntos de relaciones binarias, Profesores, con los atributos AP , y Textos, con los atributos AL . Como se trata de dos relaciones básicamente independientes, su modelado mediante un solo conjunto de relaciones ternarias con los atributos APL resulta inadecuado. (Véase en el Apartado 2.5.3 un examen más amplio de las relaciones ternarias y de las relaciones binarias.) Dada la subjetividad del diseño ER, no obstante, se puede crear una relación ternaria. Un análisis detenido de la información de las DM revelaría posteriormente el problema.

Sea R el esquema de una relación y sean X e Y subconjuntos de los atributos de R . De manera intuitiva, se dice que la **dependencia multivalorada** $X \rightarrow\!\!\! \rightarrow Y$ se cumple en R si, en todos los ejemplares legales r de R , cada valor de X está asociado con un conjunto de valores Y y ese conjunto es independiente del valor de los demás atributos.

Formalmente, si la DM $X \rightarrow\!\!\! \rightarrow Y$ se cumple en R y $Z = R - XY$, lo siguiente debe ser cierto para todos los ejemplares legales r de R :

Si $t_1 \in r$, $t_2 \in r$ y $t_1.X = t_2.X$, debe haber algún $t_3 \in r$ tal que $t_1.XY = t_3.XY$ y $t_2.Z = t_3.Z$.

La Figura 12.14 ilustra esta definición. A partir de las dos primeras tuplas y asumiendo que la DM $X \rightarrow\!\!\! \rightarrow Y$ se cumple en esta relación, se puede inferir que el ejemplar de la relación debe contener también la tercera tupla. En realidad, intercambiando los papeles de las dos primeras tuplas —tratando la primera tupla como t_2 y la segunda como t_1 — se puede deducir que la tupla t_4 debe hallarse también en el ejemplar de la relación.

X	Y	Z	
a	b_1	c_1	— tupla t_1
a	b_2	c_2	— tupla t_2
a	b_1	c_2	— tupla t_3
a	b_2	c_1	— tupla t_4

Figura 12.14 Ilustración de la definición de DM

Esta tabla sugiere otra manera de considerar las DM. Si $X \rightarrow\!\!\! \rightarrow Y$ se cumple en R , entonces $\pi_{YZ}(\sigma_{X=x}(R)) = \pi_Y(\sigma_{X=x}(R)) \times \pi_Z(\sigma_{X=x}(R))$ en todos los ejemplares legales de R , para cualquier valor x que aparezca en la columna X de R . En otras palabras, considérense grupos de tuplas de R con el mismo valor de X . En cada uno de esos grupos considérese la proyección sobre los atributos YZ . Esta proyección debe ser igual al producto cartesiano de las proyecciones sobre Y y sobre Z . Es decir, para un valor dado de X , los valores de Y y los

de Z son independientes. (A partir de esta definición es fácil ver que $X \rightarrow\!\!\! \rightarrow Y$ debe cumplirse siempre que se cumpla $X \rightarrow Y$. Si la DF $X \rightarrow Y$ se cumple, hay exactamente un valor de Y para cada valor de X , y las condiciones de la definición de la DM se cumplen trivialmente. Lo contrario no se cumple, como muestra la Figura 12.14.)

Volviendo al ejemplo *APL*, la restricción de que los textos de las asignaturas son independientes de los profesores se puede expresar como $C \rightarrow\!\!\! \rightarrow T$. En términos de la definición de las DM, esta restricción se puede leer así:

Si (hay una tupla que muestra que) A la imparte el profesor P
y (hay una tupla que muestra que) A tiene el libro L como texto,
entonces (hay una tupla que muestra que) A la imparte P y tiene como texto a L .

Dado un conjunto de DF y de DM, en general, se puede inferir que se cumplen varias DF y DM más. Un conjunto seguro y completo de reglas de inferencia consta de los tres axiomas de Armstrong y de otras cinco reglas. Tres de esas reglas sólo implican a DM:

- **Complementariedad de las DM.** Si $X \rightarrow\!\!\! \rightarrow Y$, entonces $X \rightarrow\!\!\! \rightarrow R - XY$.
- **Aumentatividad de las DM.** Si $X \rightarrow\!\!\! \rightarrow Y$ y $W \supseteq Z$, entonces $WX \rightarrow\!\!\! \rightarrow YZ$.
- **Transitividad de las DM.** Si $X \rightarrow\!\!\! \rightarrow Y$ y $Y \rightarrow\!\!\! \rightarrow Z$, entonces $X \rightarrow\!\!\! \rightarrow (Z - Y)$.

Como ejemplo del empleo de estas reglas, dado que se tiene que $A \rightarrow\!\!\! \rightarrow P$ en *APL*, la complementariedad de las DM permite inferir que también $A \rightarrow\!\!\! \rightarrow APL - AP$, es decir, $A \rightarrow\!\!\! \rightarrow L$. Las otras dos reglas relacionan las DF con las DM:

- **Réplica.** Si $X \rightarrow Y$, entonces $X \rightarrow\!\!\! \rightarrow Y$.
- **Coalescencia.** Si $X \rightarrow\!\!\! \rightarrow Y$ y hay una W tal que $W \cap Y$ está vacía, $W \rightarrow Z$ e $Y \supseteq Z$, entonces $X \rightarrow Z$.

Obsérvese que la réplica afirma que todas las DF son también DM.

12.8.2 Cuarta forma normal

La cuarta forma normal es una generalización directa de la FNBC. Sea R el esquema de una relación, X e Y subconjuntos no vacíos de los atributos de R y F un conjunto de dependencias que incluye tanto DF como DM. Se dice que R se halla en la **cuarta forma normal (4FN)** si, para toda DM $X \rightarrow\!\!\! \rightarrow Y$ que se cumple en R , es cierta una de las siguientes afirmaciones:

- $Y \subseteq X$ o $XY = R$ o
- X es una superclave.

Al leer esta definición es importante comprender que la definición de *clave* no ha cambiado —una clave debe determinar de manera única todos los atributos exclusivamente por medio de DF—. $X \rightarrow\!\!\! \rightarrow Y$ es una **DM trivial** si $Y \subseteq X \subseteq R$ o $XY = R$; estas DM se cumplen siempre.

La relación *APL* no se halla en 4FN porque $A \rightarrow\!\!\! \rightarrow P$ es una DM no trivial y A no es una clave. Se puede eliminar la redundancia resultante descomponiendo *APL* en *AP* y *AL*; cada una de estas relaciones se halla entonces en 4FN.

Para aprovechar completamente la información de las DM es necesario comprender su teoría. No obstante, el siguiente resultado, debido a Date y a Fagin identifica las condiciones—detectadas empleando únicamente información de las DF—bajo las que se puede ignorar sin riesgos la información de las DM. Es decir, emplear la información de las DM junto con la de las DF no revela ninguna redundancia. Por tanto, si se cumplen estas condiciones, ni siquiera hace falta identificar todas las DM.

Si el esquema de una relación se halla en FNBC y, como mínimo, una de sus claves consta de un solo atributo, también se halla en 4FN.

Una suposición importante se halla implícita en cualquier aplicación de este resultado: *El conjunto de DF identificado hasta ahora es realmente el conjunto de todas las DF que se cumplen en la relación*. Esta suposición es importante, ya que el resultado se basa en que la relación se halle en FNBC que, a su vez, depende del conjunto de DF que se cumplen en la relación.

Se ilustrará este punto mediante un ejemplo. Considérese el esquema de la relación $ABCD$ y supóngase la DF $A \rightarrow BCD$ y la DM $B \twoheadrightarrow C$. Considerando únicamente estas dependencias, parece que el esquema de esta relación sea un contraejemplo del resultado. La relación tiene una clave simple, parece hallarse en FNBC y, aun así, no se halla en 4FN porque $B \twoheadrightarrow C$ provoca una violación de las condiciones de la 4FN. Examinémoslo más de cerca.

B	C	A	D	
b	c_1	a_1	d_1	— tupla t_1
b	c_2	a_2	d_2	— tupla t_2
b	c_1	a_2	d_2	— tupla t_3

Figura 12.15 Tres tuplas de un ejemplar legal de $ABCD$

La Figura 12.15 muestra tres tuplas de un ejemplar de $ABCD$ que satisface la DM $B \twoheadrightarrow C$ dada. A partir de la definición de DM, dadas las tuplas t_1 y t_2 , se deduce que la tupla t_3 se debe incluir también en el ejemplar. Considérense las tuplas t_2 y t_3 . A partir de la DF $A \rightarrow BCD$ dada y del hecho de que estas tuplas tienen el mismo valor de A , se puede deducir que $c_1 = c_2$. Por tanto, puede verse que la DF $B \rightarrow C$ se debe cumplir en $ABCD$ siempre que se cumplan la DF $A \rightarrow BCD$ y la DM $B \twoheadrightarrow C$. Si se cumple que $B \rightarrow C$, la relación $ABCD$ no se halla en FNBC (a menos que otras DF hagan de B una clave).

Por tanto, el aparente contraejemplo no lo es realmente—más bien ilustra la importancia de identificar de manera correcta todas las DF que se cumplen en cada relación—. En este ejemplo $A \rightarrow BCD$ no es la única DF; la DF $B \rightarrow C$ se cumple también, pero no se identificó inicialmente. Dado un conjunto de DF y de DM, se pueden utilizar las reglas de inferencia para deducir DF (y DM) adicionalmente; para aplicar el resultado de Date-Fagin sin emplear antes las reglas de inferencia de las DM hay que estar seguro de que se han identificado todas las DF.

En resumen, el resultado de Date-Fagin ofrece una manera cómoda de comprobar si las relaciones se hallan en 4FN (sin razonar sobre las DM) cuando estamos seguros de haber identificado todas las DF. En este momento se invita al lector a repasar los ejemplos que se han estudiado en este capítulo y comprobar si hay alguna relación que no se halle en 4FN.

12.8.3 Dependencias de reunión

La reunión por dependencia es una generalización ulterior de las DM. Se dice que la **dependencia de reunión (DR)** $\bowtie \{R_1, \dots, R_n\}$ se cumple en la relación R si R_1, \dots, R_n es una descomposición por reunión sin pérdida de R .

Se puede expresar la DM $X \rightarrow\!\! \rightarrow Y$ en la relación R como la dependencia de reunión $\bowtie \{XY, X(R-Y)\}$. Por ejemplo, en la relación APL se puede expresar la DM $A \rightarrow\!\! \rightarrow P$ como la dependencia de reunión $\bowtie \{AP, AL\}$.

A diferencia de las DF y de las DM, no hay ningún conjunto de reglas de inferencia seguras y completas para las DR.

12.8.4 Quinta forma normal

Se dice que el esquema de la relación R se halla en la **quinta forma normal (5FN)** si, para todas las DR $\bowtie \{R_1, \dots, R_n\}$ que se cumplen en R , es cierta una de las siguientes afirmaciones:

- $R_i = R$ para alguna i o
- La DR está implícita en el conjunto de las DF de R en las que el primer término es una clave de R .

La segunda condición merece alguna explicación, ya que no se han presentado reglas de inferencia para las DF y las DR tomadas en conjunto. De manera intuitiva, se debe poder probar que la descomposición de R en $\{R_1, \dots, R_n\}$ es una reunión sin pérdida siempre que se cumplan las **dependencias de las claves** (DF en las que el primer término es una clave de R). La DR $\bowtie \{R_1, \dots, R_n\}$ es una **DR trivial** si $R_i = R$ para alguna i ; esas DR se cumplen siempre.

El resultado siguiente, también debido a Date y a Fagin, identifica las condiciones —nuevamente detectadas empleando sólo la información de las DF— bajo las que se puede ignorar sin riesgos la información de las DR:

Si el esquema de una relación se halla en 3FN y cada una de sus claves consta de un solo atributo, también se halla en 5FN.

Las condiciones identificadas en este resultado son suficientes, pero no necesarias, para que una relación se halle en 5FN. El resultado puede resultar muy útil en la práctica, ya que permite concluir que una relación se halla en 5FN *sin ni siquiera identificar las DM y las DR que puedan cumplirse en ella*.

12.8.5 Dependencias de inclusión

Las DM y las DR se pueden emplear para orientar el diseño de las bases de datos, como ya se ha visto, aunque sean menos frecuentes que las DF y más difíciles de reconocer y de razonar sobre ellas. Por el contrario, las dependencias de inclusión resultan muy intuitivas y bastante frecuentes. Sin embargo, suelen tener poca influencia en el diseño de las bases de datos (más allá de la fase de diseño ER).

De manera informal, una dependencia de inclusión es una declaración de la forma en que algunas columnas de una relación se hallan contenidas en otras columnas (generalmente de una segunda relación). Una restricción de clave externa es un ejemplo de dependencia de inclusión; la(s) columna(s) que hace(n) la referencia de una relación debe(n) estar contenida(s) en la(s) columna(s) de la clave principal de la relación a la que se hace referencia. Como ejemplo adicional, si R y S son dos relaciones obtenidas mediante la traducción de dos conjuntos de entidades en los que todas las entidades R son también entidades S , se tendrá una dependencia de inclusión; la proyección de R sobre los atributos de su clave genera una relación contenida en la relación obtenida por la proyección de S sobre los atributos de su clave.

Lo que hay que recordar es que no se deben dividir los grupos de atributos que participan en una dependencia de inclusión. Por ejemplo, si se tiene la dependencia de inclusión $AB \subseteq CD$, aunque se descomponga el esquema de la relación que contiene AB , se debe garantizar que, como mínimo, uno de los esquemas obtenidos en la descomposición contenga tanto A como B . En caso contrario, no se puede comprobar la dependencia de inclusión $AB \subseteq CD$ sin reconstruir la relación que contiene AB .

En la práctica, la mayor parte de las dependencias de inclusión se *basan en las claves*, es decir, sólo implican a claves. Las restricciones de clave externa son un buen ejemplo de dependencias de inclusión basadas en las claves. Los diagramas ER que implican jerarquías ES (véase el Apartado 2.4.4) también llevan a dependencias de inclusión basadas en las claves. Si todas las dependencias de inclusión se basan en las claves, rara vez habrá que preocuparse por la división de los grupos de atributos que participan en las dependencias de inclusión, ya que las descomposiciones no suelen dividir la clave principal. Téngase en cuenta, no obstante, que pasar de la 3FN a la FNBC siempre supone la división de alguna clave (no de la clave principal, en teoría), ya que la dependencia que orienta la división es de la forma $X \rightarrow A$, donde A forma parte de una clave.

12.9 ESTUDIO DE UN CASO: LA TIENDA EN INTERNET

Recuérdese del Apartado 3.8 que TiposBD se decidió por el esquema siguiente:

```
Libros(isbn: CHAR(10), título: CHAR(8), autor: CHAR(80),
       stock: INTEGER, precio: REAL, año_publicación: INTEGER)
Clientes(idc: INTEGER, nombrec: CHAR(80), dirección: CHAR(200))
Pedidos(númpedido: INTEGER, isbn: CHAR(10), idc: INTEGER,
       tarjeta: CHAR(16), cantidad: INTEGER, fecha_pedido: DATE, fecha_envío: DATE)
```

TiposBD analiza el conjunto de relaciones en búsqueda de posibles redundancias. La relación Libros sólo tiene una clave (*isbn*) y no se cumple ninguna otra dependencia funcional en esa tabla. Por tanto, Libros se halla en FNBC. La relación Clientes también tiene una única clave (*idc*) y no se cumple ninguna otra dependencia funcional en esa tabla. Por tanto, Clientes se halla también en FNBC.

Tipos BD ya ha identificado la pareja $\langle \text{númpedido}, \text{isbn} \rangle$ como clave de la tabla Pedidos. Además, dado que cada pedido lo formula un cliente en una fecha concreta con un número de tarjeta de crédito determinado, se cumplen las tres dependencias funcionales siguientes:

númpedido → *idc*, *númpedido* → *fecha_pedido* y *númpedido* → *tarjeta*

Los expertos de TiposBD concluyen que Pedidos no se halla ni siquiera en 3FN. (¿Puede el lector ver el motivo?) Deciden descomponer Pedidos en las dos relaciones siguientes:

Pedidos(*númpedido*, *idc*, *fecha_pedido*, *tarjeta* y
ListasPedidos(*númpedido*, *isbn*, *cant*, *fecha_envío*)

Las dos relaciones resultantes, Pedidos y ListasPedidos, se hallan en FNBC y la descomposición es una descomposición por reunión sin pérdida, ya que *númpedido* es clave de (la nueva relación) Pedidos. Se invita al lector a comprobar que esta descomposición también conserva las dependencias. En aras de la completitud, se dan a continuación el LDD de SQL para las relaciones Pedidos y ListasPedidos:

```
CREATE TABLE Pedidos (númpedido INTEGER,
                      idc      INTEGER,
                      fecha_pedido DATE,
                      tarjeta   CHAR(16),
                      PRIMARY KEY (númpedido),
                      FOREIGN KEY (idc) REFERENCES Clientes )
```

```
CREATE TABLE ListasPedidos ( númpedido   INTEGER,
                             isbn        CHAR(10),
                             cant       INTEGER,
                             fecha_envío DATE,
                             PRIMARY KEY (númpedido, isbn),
                             FOREIGN KEY (isbn) REFERENCES Libros)
```

La Figura 12.16 muestra un diagrama ER actualizado que refleja el nuevo diseño. Téngase en cuenta que TiposBD podría haber llegado de manera inmediata a este diagrama si hubieran hecho desde el principio de Pedidos un conjunto de entidades en lugar de un conjunto de relaciones. Pero en ese momento no comprendieron del todo los requisitos, y parecía natural modelar Pedidos como conjunto de relaciones. Este proceso iterativo de refinamiento es típico de los procesos de diseño de bases de datos en la vida real. Como TiposBD ha aprendido con el tiempo, rara vez se consigue un diseño inicial que no se modifique a medida que el proyecto avanza.

El equipo de TiposBD celebra la finalización con éxito del diseño lógico de la base de datos y del refinamiento del esquema abriendo una botella de cava y cargándose la cuenta de B&N. Tras recuperarse de la celebración, pasan a la fase de diseño físico.

12.10 PREGUNTAS DE REPASO

Las respuestas a las preguntas de repaso pueden encontrarse en los apartados indicados.

- Ilústrense la redundancia y los problemas que puede provocar. Dense ejemplos de anomalías *de inserción*, *de eliminación* y *de actualización*. ¿Pueden ayudar los valores *null* a afrontar estos problemas? ¿Son una solución completa? (**Apartado 12.1.1**)

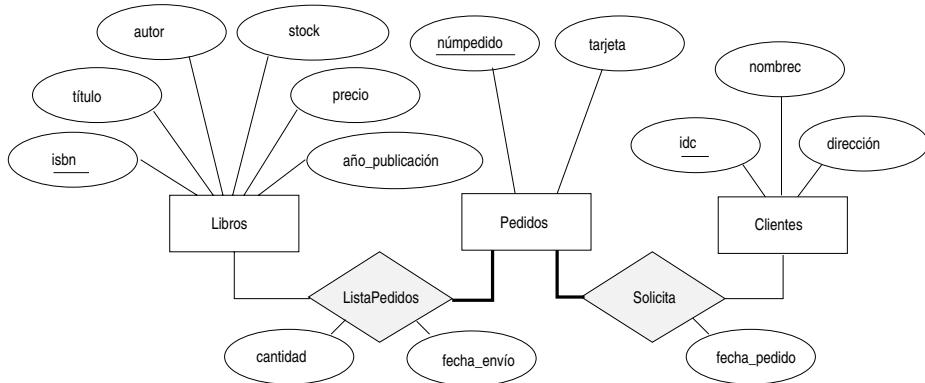


Figura 12.16 Diagrama ER que refleja el diseño final

- ¿Qué es una *descomposición* y cómo afronta la redundancia? ¿Qué problemas puede provocar el empleo de descomposiciones? (**Apartados 12.1.2 y 12.1.3**)
- Defínanse las *dependencias funcionales*. ¿Qué relación hay entre las *claves principales* y las DF? (**Apartado 12.2**)
- ¿Cuándo está la DF *d implícita* en el conjunto *D* de DF? Defínanse los *axiomas de Armstrong* y explíquese la afirmación de que “son un conjunto seguro y completo de reglas para la inferencia de DF”. (**Apartado 12.3**)
- ¿Qué es el *cierre de dependencias* D^+ de un conjunto *D* de DF? ¿Qué es el *cierre de atributos* X^+ del conjunto de atributos *X* con respecto al conjunto de DF *D*? (**Apartado 12.3**)
- Defínanse la 1FN, la 2FN, la 3FN y la FNBC. ¿Cuál es la razón de poner una relación en FNBC? ¿Cuál es la razón para la 3FN? (**Apartado 12.4**)
- ¿Cuándo se dice que la descomposición del esquema *R* en dos esquemas de relación *X* e *Y* es una descomposición *por reunión sin pérdida*? ¿Por qué es tan importante esta propiedad? Dese una condición necesaria y suficiente para comprobar si una descomposición es por reunión sin pérdida. (**Apartado 12.5.1**)
- ¿Cuándo se dice que una descomposición *conserva las dependencias*? ¿Por qué resulta útil esta propiedad? (**Apartado 12.5.2**)
- Describábase la manera de obtener una descomposición por reunión sin pérdida de una relación en FNBC. Dese un ejemplo para probar que puede que no haya ninguna descomposición en FNBC que conserve las dependencias. Ilústrese el modo en que se puede descomponer de diferentes maneras una relación dada para llegar a varias descomposiciones alternativas y analíicense las implicaciones para el diseño de bases de datos. (**Apartado 12.6.1**)
- Dese un ejemplo que ilustre el modo en que un conjunto de relaciones en FNBC puede tener redundancia aunque cada relación, por sí misma, no la tenga. (**Apartado 12.6.1**)

- ¿Qué es el *recubrimiento mínimo* de un conjunto de DF? Describase un algoritmo para calcular el recubrimiento mínimo de un conjunto de DF e ilústrese con un ejemplo. (**Apartado 12.6.2**)
- Describase la manera de adaptar el algoritmo para la descomposición por reunión sin pérdida en FNBC para obtener una descomposición por reunión sin pérdida que conserve las dependencias en 3FN. Describase el enfoque alternativo de *síntesis* para obtener esa descomposición en 3FN. Ilústrense ambos enfoques con un ejemplo. (**Apartado 12.6.2**)
- Analícese el modo en que el refinamiento de esquemas mediante el análisis de dependencias y la normalización puede mejorar los esquemas obtenidos mediante el diseño ER. (**Apartado 12.7**)
- Defínanse las *dependencias multivaloradas*, las *dependencias de reunión* y las *dependencias por inclusión*. Analícese el empleo de esas dependencias para el diseño de bases de datos. Defínanse la 4FN y la 5FN y explíquese la manera en que evitan ciertos tipos de redundancia que la FNBC no elimina. Describanse pruebas para la 4FN y para la 5FN que sólo utilicen DF. ¿Qué suposición acerca de las claves está implícita en estas pruebas? (**Apartado 12.8**)

EJERCICIOS

Ejercicio 12.1 Respóndanse brevemente las siguientes preguntas:

1. Defínase el término *dependencia funcional*.
2. ¿Por qué se denominan *triviales* algunas dependencias funcionales?
3. Dado un conjunto de DF para el esquema de la relación $R(A,B,C,D)$ con la clave principal AB bajo el que R se halla en 1FN pero no en 2FN.
4. Dese un conjunto de DF para el esquema de la relación $R(A,B,C,D)$ con la clave principal AB bajo el que R se halle en 2FN pero no en 3FN.
5. Considérese el esquema de la relación $R(A,B,C)$, que tiene la DF $B \rightarrow C$. Si A es una clave candidata de R , ¿es posible que R se halle en FNBC? En caso afirmativo, bajo qué condiciones? En caso negativo, explíquese el motivo.
6. Supóngase que se tiene el esquema de la relación $R(A,B,C)$, que representa una relación entre dos conjuntos de entidades con las claves A y B , respectivamente, y que R tiene (entre otras) las DF $A \rightarrow B$ y $B \rightarrow A$. Explíquese lo que significa este par de dependencias (es decir, lo que implican para la relación que se está modelando).

Ejercicio 12.2 Considérese la relación R con los cinco atributos $ABCDE$. Se dan las dependencias siguientes: $A \rightarrow B$, $BC \rightarrow E$ y $ED \rightarrow A$.

1. Enumérense todas las claves de R .
2. ¿Se halla R en 3FN?
3. ¿Se halla R en FNBC?

Ejercicio 12.3 Considérese la relación de la Figura 12.17.

1. Enumérense todas las dependencias funcionales que satisface este ejemplar de la relación.
2. Supóngase que el valor del atributo Z del último registro de la relación se cambia de z_3 a z_2 . Enumérense ahora todas las dependencias funcionales que satisface este ejemplar de la relación.

X	Y	Z
x_1	y_1	z_1
x_1	y_1	z_2
x_2	y_1	z_1
x_2	y_1	z_3

Figura 12.17 Relación para el Ejercicio 12.3**Ejercicio 12.4** Supóngase que se tiene una relación con los atributos $ABCD$.

1. Supóngase que ningún registro tiene valores NULL. Escríbase una consulta de SQL que compruebe si se cumple la dependencia funcional $A \rightarrow B$.
2. Supóngase una vez más que ningún registro tiene valores NULL. Escríbase un aserto de SQL que haga que se cumpla la dependencia funcional $A \rightarrow B$.
3. Supóngase ahora que los registros pueden tener valores NULL. Repítanse las dos preguntas anteriores con esta suposición.

Ejercicio 12.5 Considérese el siguiente conjunto de relaciones y de dependencias. Supóngase que cada relación se obtiene mediante la descomposición de una relación con los atributos $ABCDEFGHI$ y que las dependencias conocidas de la relación $ABCDEFGHI$ se indican para cada pregunta. (Las preguntas son independientes entre sí, evidentemente, ya que las dependencias dadas de $ABCDEFGHI$ son diferentes.) Para cada (sub)relación: (a) Indicar la forma normal más fuerte en la que se halla esa relación. (b) Si no se halla en FNBC, descompóngase en un conjunto de relaciones en FNBC.

1. $R1(A,C,B,D,E)$, $A \rightarrow B$, $C \rightarrow D$
2. $R2(A,B,F)$, $AC \rightarrow E$, $B \rightarrow F$
3. $R3(A,D)$, $D \rightarrow G$, $G \rightarrow H$
4. $R4(D,C,H,G)$, $A \rightarrow I$, $I \rightarrow A$
5. $R5(A,I,C,E)$

Ejercicio 12.6 Supóngase que se tienen las tres tuplas siguientes de un ejemplar legal del esquema de una relación E con tres atributos ABC (citados en orden): (1,2,3), (4,2,3) y (5,3,3).

1. ¿Cuál de las dependencias siguientes se puede inferir que *no* se cumple en el esquema E ?
 - (a) $A \rightarrow B$, (b) $BC \rightarrow A$, (c) $B \rightarrow C$
2. ¿Se puede identificar alguna dependencia que se cumpla en E ?

Ejercicio 12.7 Supóngase que se da la relación R con los cuatro atributos $ABCD$. Para cada uno de los conjuntos siguientes de DF, suponiendo que se trata de las únicas dependencias que se cumplen para R , hágase lo siguiente: (a) Identificar la(s) clave(s) candidata(s) de R . (b) Identificar la mejor forma normal que satisface R (1FN, 2FN, 3FN o FNBC). (c) Si R no se halla en FNBC, descompóngase en un conjunto de relaciones en FNBC que conserven las dependencias.

1. $C \rightarrow D$, $C \rightarrow A$, $B \rightarrow C$
2. $B \rightarrow C$, $D \rightarrow A$
3. $ABC \rightarrow D$, $D \rightarrow A$
4. $A \rightarrow B$, $BC \rightarrow D$, $A \rightarrow C$
5. $AB \rightarrow C$, $AB \rightarrow D$, $C \rightarrow A$, $D \rightarrow B$

Ejercicio 12.8 Considérese el conjunto de atributos $R = ABCFGH$ y el conjunto de DF $D = \{AB \rightarrow C, AC \rightarrow B, AF \rightarrow E, B \rightarrow F, BC \rightarrow A, E \rightarrow G\}$.

1. Para cada uno de los conjuntos de atributos siguientes, hágase lo que se indica: (i) Calcular el conjunto de dependencias que se cumplen en el conjunto y escribir un recubrimiento mínimo. (ii) Nombrar la forma normal más potente que no viola la relación que contiene estos atributos. (iii) Descomponerlo en un conjunto de relaciones en FNBC, si no se halla ya en esa forma normal.
 - (a) ABC , (b) $ABCF$, (c) $ABCEG$, (d) $FCEGH$, (e) $ACEH$
2. ¿Cuál de las siguientes descomposiciones de $R = ABCFGH$, con el mismo conjunto de dependencias D , (a) conserva las dependencias? (b) es una descomposición por reunión sin pérdida?
 - (a) $\{AB, BC, ABEF, EG\}$
 - (b) $\{ABC, ACEF, AFG\}$

Ejercicio 12.9 Supongamos que R se descompone en R_1, R_2, \dots, R_n . Sea D un conjunto de DF de R .

1. Definase lo que significa para D *conservarse* en el conjunto de relaciones descompuestas.
2. Describase un algoritmo polinómico en el tiempo que compruebe la conservación de las dependencias.
3. Proyectar las DF definidas en el conjunto de atributos X sobre un subconjunto de atributos Y exige que se considere el cierre de las DF. Dese un ejemplo en el que considerar el cierre sea importante para comprobar la conservación de las dependencias, es decir, que considerar sólo las DF dadas dé resultados incorrectos.

Ejercicio 12.10 Supóngase la relación $R(A,B,C,D)$. Para cada uno de los siguientes conjuntos de DF, suponiendo que son las únicas dependencias que se cumplen en R , hágase lo que se indica: (a) Identificar la(s) clave(s) candidata(s) de R . (b) Indicar si la descomposición de R en relaciones más pequeñas propuesta es una buena descomposición y explicar brevemente el motivo.

1. $B \rightarrow C, D \rightarrow A$; se descompone en BC y AD .
2. $AB \rightarrow C, C \rightarrow A, C \rightarrow D$; se descompone en ACD y BC .
3. $A \rightarrow BC, C \rightarrow AD$; se descompone en ABC y AD .
4. $A \rightarrow B, B \rightarrow C, C \rightarrow D$; se descompone en AB y ACD .
5. $A \rightarrow B, B \rightarrow C, C \rightarrow D$; se descompone en AB, AD y CD .

Ejercicio 12.11 Considérese la relación R que tiene tres atributos ABC . Se descompone en las relaciones R_1 , con los atributos AB , y R_2 , con los atributos, BC .

1. Formúlese la definición de descomposición por reunión sin pérdida con respecto a este ejemplo. Respóndase concisamente a esta pregunta escribiendo la ecuación del álgebra relacional que implica a R, R_1 y R_2 .
2. Supóngase que $B \rightarrow C$. ¿Es la descomposición de R en R_1 y R_2 por reunión sin pérdida? Contrástese la respuesta con la observación de que no se cumple ninguna de las DF, ni $R_1 \cap R_2 \rightarrow R_1$ ni $R_1 \cap R_2 \rightarrow R_2$, a la vista de la prueba sencilla que ofrece una condición necesaria y suficiente para la descomposición por reunión sin pérdida en dos relaciones del Apartado 15.6.1.
3. Si se dan las siguientes ejemplares de R_1 y de R_2 , ¿qué se puede decir del ejemplar de R de la que se han obtenido? Respóndase esta respuesta indicando las tuplas que se hallan con seguridad en R y las que puede que se hallen en R .

$$\begin{aligned} \text{Ejemplar de } R_1 &= \{(5,1), (6,1)\} \\ \text{Ejemplar de } R_2 &= \{(1,8), (1,9)\} \end{aligned}$$

¿Se puede decir con seguridad que el atributo B es o no es clave de R ?

Ejercicio 12.12 Supóngase que se dispone de las cuatro tuplas siguientes de la relación S de tres atributos (ABC): $(1,2,3), (4,2,3), (5,3,3), (5,3,4)$. ¿Cuáles de las siguientes dependencias funcionales (\rightarrow) y multivaloradas ($\rightarrow\rightarrow$) se puede inferir que no se cumplen en la relación S ?

1. $A \rightarrow B$
2. $A \nrightarrow B$
3. $BC \rightarrow A$
4. $BC \nrightarrow A$
5. $B \rightarrow C$
6. $B \nrightarrow C$

Ejercicio 12.13 Considérese la relación R con los cinco atributos $ABCDE$.

1. Para cada una de las siguientes ejemplares de R , indíquese si viola (a) la DF $BC \rightarrow D$ y (b) la DM $BC \nrightarrow D$:
 - (a) $\{ \}$ (es decir, relación vacía)
 - (b) $\{(a,2,3,4,5), (2,a,3,5,5)\}$
 - (c) $\{(a,2,3,4,5), (2,a,3,5,5), (a,2,3,4,6)\}$
 - (d) $\{(a,2,3,4,5), (2,a,3,4,5), (a,2,3,6,5)\}$
 - (e) $\{(a,2,3,4,5), (2,a,3,7,5), (a,2,3,4,6)\}$
 - (f) $\{(a,2,3,4,5), (2,a,3,4,5), (a,2,3,6,5), (a,2,3,6,6)\}$
 - (g) $\{(a,2,3,4,5), (a,2,3,6,5), (a,2,3,6,6), (a,2,3,4,6)\}$
2. Si todos los ejemplares anteriores de R son legales, ¿qué se puede decir de la DF $A \rightarrow B$?

Ejercicio 12.14 Las DR se deben al hecho de que a veces una relación que no se puede descomponer en dos relaciones por reunión sin pérdida más pequeñas sí puede descomponerse así en tres o más relaciones. Por ejemplo, la relación con los atributos *proveedor*, *repuesto* y *proyecto*, denotada PRY , sin DF ni DM. Se cumple la DR $\bowtie \{SP, PJ, JS\}$.

A partir de la DR, el conjunto de esquemas de la relación PR , RY e YP es una descomposición por reunión sin pérdida de PRY . Constrúyase un ejemplar de PRY para ilustrar que no basta con dos de estos esquemas.

Ejercicio 12.15 Respóndase a las preguntas siguientes.

1. Pruébese que el algoritmo de la Figura 12.4 calcula correctamente el cierre de los atributos del conjunto inicial de atributos X .
2. Describábase un algoritmo lineal en el tiempo (en el tamaño del conjunto de DF, donde el tamaño de cada DF es el número de atributos implicados) para hallar el cierre de los atributos con respecto a un conjunto de DF. Pruébese que ese algoritmo calcula correctamente el cierre de los atributos del conjunto inicial de atributos.

Ejercicio 12.16 Se dice que la DF $X \rightarrow Y$ es *simple* si Y es un solo atributo.

1. Sustítuyase la DF $AB \rightarrow CD$ por el conjunto equivalente mínimo de DF simples.
2. Pruébese que todas las DF $X \rightarrow Y$ del conjunto de DF D se pueden sustituir por un un conjunto de DF simples tales que D^+ sea igual al cierre del nuevo conjunto de DF.

Ejercicio 12.17 Pruébese que los axiomas de Armstrong son seguros y completos para la inferencia de DF. Es decir, muéstrese que la aplicación repetida de estos axiomas sobre el conjunto D de DF produce exactamente las dependencias de D^+ .

Ejercicio 12.18 Considérese la relación R con los atributos $ABCDE$. Sean las siguientes DF: $A \rightarrow BC$, $BC \rightarrow E$ y $E \rightarrow DA$. De manera parecida, sea S una relación con los atributos $ABCDE$ y las siguientes DF: $A \rightarrow BC$, $B \rightarrow E$ y $E \rightarrow DA$. (Sólo la segunda dependencia se diferencia de las que se cumplen en R .) No se sabe si se cumple alguna dependencia (por reunión) más.

1. ¿Se halla R en FNBC?
2. ¿Se halla R en 4FN?
3. ¿Se halla R en 5FN?
4. ¿Se halla S en FNBC?
5. ¿Se halla S en 4FN?
6. ¿Se halla S en 5FN?

Ejercicio 12.19 Sea R el esquema de una relación con el conjunto D de DF. Pruébese que la descomposición de R en R_1 y R_2 se realiza por reunión sin pérdida y sólo si D^+ contiene $R_1 \cap R_2 \rightarrow R_1$ o $R_1 \cap R_2 \rightarrow R_2$.

Ejercicio 12.20 Considérese el esquema R con las DF D que se descompone en esquemas con los atributos X e Y . Demuéstrese que esta descomposición conserva las dependencias si $D \subseteq (D_X \cup D_Y)^+$.

Ejercicio 12.21 Pruébese que la optimización del algoritmo para la descomposición por reunión sin pérdida que conserva las dependencias en relaciones en 3FN (Apartado 12.6.2) es correcto.

Ejercicio 12.22 Pruébese que el algoritmo de síntesis de la 3FN produce una descomposición por reunión sin pérdida de la relación que contiene todos los atributos originales.

Ejercicio 12.23 Pruébese que la DM $X \twoheadrightarrow Y$ de la relación R se puede expresar como la dependencia por reunión $\bowtie \{XY, X(R - Y)\}$.

Ejercicio 12.24 Pruébese que, si R sólo tiene una clave, se halla en FNBC si y sólo si se halla en 3FN.

Ejercicio 12.25 Pruébese que, si R se halla en 3FN y todas las claves son simples, R se halla en FNBC.

Ejercicio 12.26 Pruébense estas afirmaciones:

1. Si el esquema de una relación se halla en FNBC y, como mínimo, una de sus claves consta de un único atributo, también se halla en 4FN.
2. Si el esquema de una relación se halla en 3FN y cada clave tiene un único atributo, también se halla en 5FN.

Ejercicio 12.27 Dese un algoritmo para comprobar si el esquema de una relación se halla en FNBC. El algoritmo debe ser polinómico en el tamaño del conjunto de DF dadas. (El *tamaño* es la suma para todas las DF del número de atributos que aparecen en cada DF.) ¿Hay algún algoritmo polinómico para comprobar si el esquema de una relación se halla en 3FN?

Ejercicio 12.28 Pruébese que el algoritmo para descomponer el esquema de una relación con una serie de DF en un conjunto de esquemas de relaciones en FNBC de la manera descrita en el Apartado 12.6.1 es correcto (es decir, genera un conjunto de relaciones en FNBC y es una descomposición por reunión sin pérdida) y termina.

NOTAS BIBLIOGRÁFICAS

Entre las presentaciones en libros de texto de la teoría de las dependencias y de su empleo en el diseño de bases de datos están [1, 29, 309, 315, 455]. Entre los buenos artículos de investigación sobre el tema figuran [461, 265]. Las DF se introdujeron en [119], junto con el concepto de 3FN, y los axiomas para la inferencia de DF se presentaron en [23]. La FNBC se introdujo en [120]. El concepto de ejemplar legal de una relación y de cumplimiento de las dependencias se estudiaron formalmente en [213]. Las DF se generalizaron a los modelos semánticos de datos en [467].

En [306] se demuestra que hallar una clave es NP completo. Las descomposiciones por reunión sin pérdida se estudiaron en [17, 310, 377]. Las descomposiciones que conservan las dependencias se estudiaron en [49]. [52] introdujo los recubrimientos mínimos. La descomposición en 3FN se estudia en [52, 58] y la descomposición en FNBC se aborda en [450]. [264] muestra que comprobar si una relación se halla en 3FN es NP completo. [163] introdujo la 4FN y examinó la descomposición en 4FN. Fagin introdujo otras formas normales en [164] (forma normal de proyección-reunión) y [165] (forma normal de dominio-clave). A diferencia del extenso estudio de las descomposiciones verticales, ha habido relativamente poca investigación de las descomposiciones horizontales. [141] investiga las descomposiciones horizontales.

Las DM fueron descubiertas de manera independiente por Delobel [143], Fagin [163] y Zaniolo [480]. Los axiomas para las DF y las DM se presentaron en [48]. [357] muestra que no hay ninguna axiomatización para las DR, aunque [397] ofrece una axiomatización para una clase más general de dependencias. Las condiciones suficientes para la 4FN y para la 5FN en términos de DF que se estudiaron en el Apartado 12.8 provienen de [138]. [314, 315] describen un enfoque del diseño de bases de datos que emplea la información sobre las dependencias para crear muestras de ejemplares de las relaciones.



13

DISEÑO FÍSICO Y AJUSTE DE BASES DE DATOS

- ¿Qué es el diseño físico de bases de datos?
- ¿Qué es la carga de trabajo de las consultas?
- ¿Cómo se escogen los índices? ¿De qué herramientas se dispone?
- ¿Qué es la coagrupación y cómo se utiliza?
- ¿Cuáles son las opciones de ajuste de las bases de datos?
- ¿Cómo se ajustan las consultas y las vistas?
- ¿Cuál es el impacto de la concurrencia en el rendimiento?
- ¿Cómo se pueden reducir la contención por bloqueos y los puntos calientes?
- ¿Cuáles son las pruebas de homologación de bases de datos más populares y cómo se utilizan?
- ▶ **Conceptos fundamentales:** diseño físico de bases de datos, ajuste de bases de datos, carga de trabajo, coagrupación, ajuste de índices, asistente para ajuste, configuración de los índices, puntos calientes, contención por bloqueos, pruebas de homologación de bases de datos, transacciones por segundo.

Consejo a un cliente que se ha quejado de que la lluvia se filtra por el tejado hasta la mesa del comedor: "Mueva la mesa."

— Frank Lloyd Wright, Arquitecto

El rendimiento de los SGBD en consultas planteadas habitualmente y en operaciones de actualización típicas es la medida definitiva del diseño de las bases de datos. El administrador de bases de datos puede mejorar el rendimiento identificando los cuellos de botella del rendimiento y ajustando algunos parámetros del SGBD (por ejemplo, el tamaño del grupo de memorias intermedias o la frecuencia de los puntos de control) o añadiendo infraestructuras que eliminén esos cuellos de botella. El primer paso para lograr un buen rendimiento, no

obstante, es tomar buenas decisiones en el diseño de la base de datos, que es el centro de atención de este capítulo.

Tras el diseño de los esquemas *conceptual* y *externo*, es decir, la creación de una serie de relaciones y de vistas junto con un conjunto de restricciones de integridad, hay que abordar los objetivos de rendimiento mediante el **diseño físico de la base de datos**, en el que se diseña el esquema *físico*. A medida que evolucionan las necesidades de los usuarios, suele ser necesario **ajustar**, o configurar, todos los aspectos del diseño de la base de datos para conseguir un buen rendimiento.

Este capítulo está organizado de la manera siguiente. En el Apartado 13.1 se ofrece una visión general del diseño físico de las bases de datos y de su ajuste. Las decisiones de diseño físico más importantes afectan a la elección de los índices. En el Apartado 13.2 se ofrecen directrices para decidir los índices que se deben crear. Estas directrices se ilustran mediante varios ejemplos y se desarrollan más en el Apartado 13.3. En el Apartado 13.4 se examina detenidamente el importante asunto de las agrupaciones; se examina la manera de escoger índices agrupados y si las tuplas de relaciones diferentes se deben almacenar cerca unas de otras (opción apoyada por algunos SGBD). En el Apartado 13.5 se pone énfasis en el modo en que un índice bien elegido puede permitir responder algunas consultas sin necesidad de examinar los propios registros de datos. El Apartado 13.6 examina las herramientas que pueden ayudar a los DBA a seleccionar índices de manera automática.

En el Apartado 13.7 se resumen los principales problemas del ajuste de las bases de datos. Además de ajustar los índices, puede que haya que ajustar el esquema conceptual y consultas y definiciones de vistas que se empleen con frecuencia. En el Apartado 13.8 se examina el modo de refinar el esquema conceptual y, en el Apartado 13.9, el modo de refinar las consultas y las definiciones de vistas. En el Apartado 13.10 se examina brevemente la repercusión en el rendimiento del acceso concurrente. En el Apartado 13.11 se ilustra el ajuste del ejemplo de la tienda de Internet. Se concluye el capítulo con un breve examen de las pruebas de homologación de SGBD en el Apartado 13.12; las pruebas de homologación ayudan a evaluar el rendimiento de productos de SGBD alternativos.

13.1 INTRODUCCIÓN AL DISEÑO FÍSICO DE BASES DE DATOS

Al igual que todos los demás aspectos del diseño de bases de datos, el diseño físico se debe guiar por la naturaleza de los datos y por el fin al que se destinan. En concreto, es importante comprender la **carga de trabajo** habitual que debe soportar la base de datos; la carga de trabajo consiste en una mezcla de consultas y de actualizaciones. Los usuarios también tienen ciertas exigencias sobre la velocidad a la que deben ejecutarse determinadas consultas o actualizaciones o sobre el número de transacciones que se deben procesar por segundo. La descripción de la carga de trabajo y de las exigencias de rendimiento de los usuarios son la base sobre la que hay que tomar varias decisiones durante el diseño físico de las bases de datos.

Para crear un buen diseño de una base de datos y ajustar el sistema para mejorar el rendimiento en respuesta de las variables exigencias de los usuarios, los diseñadores deben comprender la manera en que funcionan los SGBD, especialmente las técnicas de indexación

Identificación de los puntos calientes para el rendimiento. Todos los sistemas comerciales ofrecen un conjunto de herramientas para el control de una amplia gama de parámetros del sistema. Estas herramientas, empleadas adecuadamente, pueden ayudar a identificar los puntos calientes para el rendimiento y sugerir aspectos del diseño de la base de datos y del código de la aplicación que hay que ajustar para mejorarlo. Por ejemplo, se puede pedir al SGBD que vigile la ejecución de la base de datos durante un periodo de tiempo determinado y que informe del número de búsquedas agrupadas, cursores abiertos, peticiones de bloqueo, puntos de control, exámenes de memorias intermedias, tiempo de espera medio de los bloqueos y otras muchas estadísticas similares que dan una visión detallada de una *instantánea* del sistema en funcionamiento. En Oracle se puede generar un informe que contenga esta información ejecutando un archivo de secuencia de comandos denominado UTLBSTAT.SQL para que comience la vigilancia y el archivo de secuencia de comandos UTLBSTAT.SQL para que concluya. El catálogo del sistema contiene detalles sobre el tamaño de las tablas, la distribución de los valores en las claves de los índices y otros parecidos. El plan generado por el SGBD para una consulta dada se puede ver en una presentación gráfica que muestre el coste estimado de cada operador del plan. Aunque los detalles son específicos de cada fabricante, todos los productos de SGBD principales del mercado ofrecen hoy en día un conjunto de herramientas de este tipo.

y de procesamiento de consultas que soporta cada SGBD. Si se espera que muchos usuarios tengan acceso a la base de datos de manera concurrente o se trata de una *base de datos distribuida*, la labor se vuelve más complicada y entran en juego otras características de los SGBD. La repercusión de la concurrencia en el diseño de bases de datos se examina en el Apartado 13.10.

13.1.1 Cargas de trabajo en las bases de datos

La clave de un buen diseño físico radica en llegar a una descripción precisa de la carga de trabajo esperada. La **descripción de la carga de trabajo** incluye los aspectos siguientes:

1. Una lista de consultas (con su frecuencia, en forma de proporción del total de consultas / actualizaciones).
2. Una lista de actualizaciones y de sus frecuencias.
3. Objetivos de rendimiento para cada tipo de consulta y de actualización.

Para cada consulta de la carga de trabajo hay que identificar:

- Las relaciones a las que se tiene acceso.
- Los atributos que se conservan (en la cláusula SELECT).
- Los atributos para los que se expresan condiciones de selección o de reunión (en la cláusula WHERE) y lo selectivas que puedan llegar a ser esas condiciones.

De manera parecida, para cada actualización de la carga de trabajo hay que identificar:

- Los atributos para los que se expresan condiciones de selección o de reunión (en la cláusula WHERE) y lo selectivas que puedan llegar a ser esas condiciones.
- El tipo de actualización (INSERT, DELETE o UPDATE) y la relación que se actualiza.
- Para las órdenes UPDATE, los campos que modifica esa actualización.

Recuérdese que las consultas y las actualizaciones suelen tener parámetros; por ejemplo, cada operación de débito o de crédito implica un número de cuenta concreto. El valor de esos parámetros determina la selectividad de las condiciones de selección y de reunión.

Las actualizaciones tienen un componente de consulta que se utiliza para buscar las tuplas objetivo. Este componente puede aprovecharse de un buen diseño físico y de la presencia de índices. Por otro lado, las actualizaciones suelen exigir trabajo extra para mantener los índices de los atributos que modifican. Por tanto, aunque las consultas sólo se pueden aprovechar de la presencia de índices, los índices tanto pueden acelerar como frenar una actualización dada. Los diseñadores deben tener en cuenta este equilibrio al crear los índices.

13.1.2 Diseño físico y decisiones de ajuste

Entre las decisiones importantes que se adoptan durante el diseño físico y el ajuste de las bases de datos están las siguientes:

1. *Elección de los índices a crear.*

- Las relaciones que se van a indexar y el campo o la combinación de campos que se va a elegir como claves de búsqueda del índice.
- Para cada índice, si se debe agrupar o no.

2. *Ajuste del esquema conceptual.*

- *Esquemas normalizados alternativos.* Suele haber más de una manera de descomponer un esquema en la forma normal deseada (FNBC o 3FN). Se puede realizar la elección según criterios de rendimiento.
- *Desnormalización.* Puede que se desee considerar de nuevo las descomposiciones del esquema llevadas a cabo con vistas a la normalización durante el proceso de diseño del esquema conceptual con objeto de mejorar el rendimiento de las consultas que implican a los atributos de varias relaciones descompuestas previamente.
- *Particiones verticales.* En determinadas circunstancias puede que se desee descomponer aún más las relaciones para mejorar el rendimiento de las consultas que sólo implican a unos cuantos atributos.
- *Vistas.* Puede que se desee añadir algunas vistas para ocultar a los usuarios los cambios en el esquema conceptual.

3. *Ajuste de consultas y de transacciones.* Tal vez las consultas y las transacciones que se ejecutan con frecuencia se puedan volver a formular de modo que se ejecuten más rápido.

En las bases de datos paralelas o distribuidas hay más opciones a considerar, como la división de las relaciones entre diferentes emplazamientos o el almacenamiento de copias de las relaciones en varios sitios.

13.1.3 Necesidad del ajuste de las bases de datos

Puede que la información precisa y detallada sobre la carga de trabajo sea difícil de obtener mientras se realiza el diseño inicial del sistema. Por tanto, es importante ajustar las bases de datos después de diseñarlas e implantarlas —hay que refinar el diseño inicial a la vista de las pautas de uso reales con objeto de obtener el máximo rendimiento posible—.

La distinción entre diseño y ajuste de las bases de datos es algo arbitraria. Se podría considerar concluido el proceso de diseño una vez esbozado el primer esquema conceptual y adoptado un conjunto de decisiones sobre indexación y agrupación. Los cambios posteriores del esquema conceptual o de los índices, por ejemplo, se podrían considerar como ajustes. Por otro lado, se podría considerar que cierto refinamiento del esquema conceptual (y las decisiones de diseño físico afectadas por ese refinamiento) forma parte del proceso de diseño físico.

No es muy importante dónde se trace la línea entre el diseño y el ajuste, y aquí nos limitaremos a examinar los aspectos de la selección de los índices y del ajuste de las bases de datos sin tener en cuenta cuándo se lleva a cabo ese ajuste.

13.2 DIRETRICES PARA LA SELECCIÓN DE ÍNDICES

Al tomar en consideración los índices a crear se comienza con la lista de consultas (incluidas las que aparecen como parte de las operaciones de actualización). Evidentemente, sólo hace falta considerar como candidatas a la indexación las relaciones a las que tiene acceso alguna consulta, y la elección de los candidatos a índice se guía por las condiciones que aparecen en las cláusulas `WHERE` de las consultas de la carga de trabajo. La presencia de índices adecuados puede mejorar de manera significativa los planes de evaluación las consultas.

Un enfoque de la selección de índices es considerar por orden las consultas más importantes y, para cada una de ellas, determinar el plan que seleccionaría el optimizador dados los índices presentes en la lista de índices (que se van a crear). Luego se considera si se puede conseguir un plan sustancialmente mejor añadiendo más índices; de ser así, esos índices adicionales son candidatos a la inclusión en la lista de índices. En general, las recuperaciones de rangos se ven favorecidas por los índices de árboles B+, mientras que las recuperaciones de coincidencias exactas se ven favorecidas por los índices asociativos. La agrupación favorece a las consultas por rangos, y también a las consultas de coincidencias exactas si varias entradas de datos contienen el mismo valor de la clave.

Antes de añadir un índice a la lista, no obstante, hay que tener en cuenta las consecuencias de tener este índice con respecto a las actualizaciones por la carga de trabajo. Como ya se ha señalado, aunque los índices pueden acelerar el componente de consulta de las actualizaciones, hay que actualizar todos los índices de los atributos actualizados —de *cualquier* atributo, en el caso de las inserciones y eliminaciones— siempre que se modifique el valor de esos

atributos. Por tanto, a veces se debe considerar el equilibrio entre frenar algunas operaciones de actualización de la carga de trabajo y acelerar algunas consultas.

Evidentemente escoger un buen conjunto de índices para una carga de trabajo dada exige comprender las técnicas de indexación disponibles y del funcionamiento del optimizador de consultas. Las siguientes directrices para la selección de índices resumen nuestro análisis:

Creación de índices (Directriz 1). Las cosas obvias son a menudo las más importantes. No se debe crear un índice a menos que alguna consulta —incluidos los componentes de consulta de las actualizaciones— se vea favorecida por ello. Siempre que sea posible se deben escoger índices que aceleran más de una consulta.

Elección de la clave de búsqueda (Directriz 2). Los atributos que aparecen en las cláusulas `WHERE` son candidatos a la indexación.

- Las condiciones de selección de coincidencias perfectas sugieren que se considere un índice para los atributos seleccionados; a ser posible, un índice asociativo.
- Las selecciones por rangos sugieren que se consideran índices de árboles B+ (o ISAM) para los atributos seleccionados. Los índices de árboles B+ suelen ser preferibles a los índices ISAM. Puede que merezca tomar en consideración un índice ISAM si la relación se actualiza rara vez, pero se da por supuesto que siempre se eligen los índices de árboles B+ frente a los índices ISAM en aras de la sencillez.

Claves de búsqueda para varios atributos (Directriz 3). Se deben tomar en consideración los índices con claves de búsqueda para varios atributos en las dos situaciones siguientes:

- Una cláusula `WHERE` incluye condiciones para más de un atributo de una relación.
- Habilitan estrategias de evaluación basadas únicamente en los índices (por ejemplo, se puede evitar tener acceso a la relación) para las consultas importantes. (Esta situación puede hacer que los atributos estén en la clave de búsqueda aunque no aparezcan en las cláusulas `WHERE`.)

Al crear índices para claves de búsqueda con varios atributos, si se esperan consultas por rangos hay que tener cuidado al ordenar los atributos de la clave de búsqueda para que coincidan con las consultas.

Agrupaciones (Directriz 4). Se puede agrupar como máximo un índice de cada relación, y el agrupamiento afecta enormemente al rendimiento; por tanto, la selección del índice agrupado es importante.

- Como regla general, es probable que las consultas por rangos sean las que más ventajas obtengan de la agrupación. Si se plantean varias consultas por rangos sobre una misma relación, que implican diferentes conjuntos de atributos, se deben tomar en consideración la selectividad de las consultas y su frecuencia relativa en la carga de trabajo a la hora de decidir el índice que se debe agrupar.
- Si un índice permite una estrategia de evaluación basada únicamente en los índices para la consulta que se pretende que acelere, no hace falta que ese índice esté agrupado. (La agrupación sólo tiene importancia cuando el índice se emplea para recuperar tuplas de la relación subyacente.)

Índices asociativos e índices de árbol (Directriz 5). Suelen ser preferibles los índices de árbol B+ porque soportan las consultas por rangos y las consultas de igualdad. Los índices asociativos son mejores en las situaciones siguientes:

- Se pretende que el índice soporte las reuniones de bucles anidados de índices; la relación indexada es la más interna y la clave de búsqueda incluye las columnas de la reunión. En este caso, se multiplica la pequeña mejora que suponen los índices asociativos frente a los de árboles B+ para las selecciones de igualdad, ya que se genera una selección de igualdad para cada tupla de la relación externa.
- Hay una consulta de igualdad muy importante que implica a los atributos de la clave de búsqueda, pero ninguna consulta por rango que lo haga.

Compensación del coste de mantenimiento de los índices (Directriz 6). Tras elaborar una “lista de deseos” de los índices que se van a crear hay que tomar en consideración las consecuencias de cada índice para las actualizaciones de la carga de trabajo.

- Si el mantenimiento de un índice hace más lentas las operaciones de actualización habituales hay que considerar la posibilidad de descartarlo.
- Hay que tener presente, no obstante, que añadir un índice también puede acelerar una operación de actualización dada. Por ejemplo, el índice de los identificadores de los empleados puede acelerar la operación de aumento de sueldo de un empleado concreto (especificado mediante su ID).

13.3 EJEMPLOS BÁSICOS DE SELECCIÓN DE ÍNDICES

Los ejemplos siguientes ilustran la manera de seleccionar índices durante el diseño de las bases de datos, continuando así el análisis que se inició en el Capítulo 9, centrado en la selección de índices para las consultas a una sola tabla. Los esquemas empleados en los ejemplos no se describen con detalle; en general, contienen los atributos que se mencionan en las consultas. Se ofrece información adicional cuando resulta necesaria.

Comencemos con una consulta sencilla:

```
SELECT E.nombree, D.director
  FROM Empleados E, Departamentos D
 WHERE D.nombred='Juguetes' AND E.númd=D.númd
```

Las relaciones mencionadas en la consulta son Empleados y Departamentos, y las dos condiciones de la cláusula WHERE implican igualdades. Las directrices anteriores sugieren que se construyan índices asociativos para los atributos implicados. Parece claro que se debe construir un índice asociativo para el atributo *nombred* de Departamentos. Pero considérese la igualdad *E.númd=D.númd*. ¿Se debe construir un índice (asociativo, por supuesto) para el atributo *númd* de Departamentos o para el de Empleados (o para los dos)? De manera intuitiva se desea recuperar las tuplas de Departamentos mediante el índice para *nombred*,

ya que pocas tuplas van a poder satisfacer la selección de igualdad $D.nombred = 'Juguetes'$ ¹. Para cada una de las tuplas de Departamento que cumple la condición hay que encontrar tuplas de empleados coincidentes mediante el índice para el atributo $númd$ de Empleados. Por lo tanto, se debe crear un índice para el campo $númd$ de Empleados. (Obsérvese que no se consigue nada construyendo un índice adicional para el campo $númd$ de Departamentos porque las tuplas de Departamentos se recuperan mediante el índice para $nombred$.)

La elección de los índices estuvo orientada por el plan de evaluación de consultas que se deseaba utilizar. Esta consideración de un posible plan de evaluación es frecuente a la hora de tomar decisiones para el diseño físico. La comprensión de la optimización de consultas resulta muy útil para el diseño físico. El plan deseado para esta consulta se puede ver en la Figura 13.1.

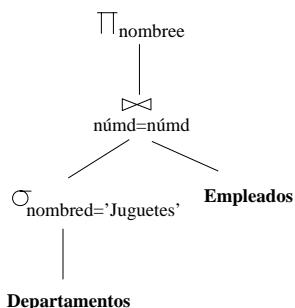


Figura 13.1 Un plan de evaluación de consultas deseable

Como variante de esta consulta, supóngase que se modifica la cláusula WHERE para que quede como WHERE $D.nombred = 'Juguetes'$ AND $E.númd = D.númd$ AND $E.edad = 25$. Consideremos planes de evaluación alternativos. Un buen plan es recuperar las tuplas de Departamentos que cumplen la selección de $nombred$ y las tuplas coincidentes de Empleados mediante un índice para el campo $númd$; se aplica luego la selección para $edad$ sobre la marcha. Sin embargo, a diferencia de la variante anterior de esta consulta, no hace falta realmente tener un índice para el campo $númd$ de Empleados si se tiene un índice para $edad$. En este caso se pueden recuperar tuplas de Departamentos que cumplan la selección para $nombred$ (mediante el índice para $nombred$, igual que antes), recuperar tuplas de Empleados que cumplan la selección para $edad$ mediante el índice para ese mismo campo y reunir esos conjuntos de tuplas. Dado que los conjuntos de tuplas que se reúnen son pequeños, caben en memoria y el método de reunión que se emplee es irrelevante. Es probable que este plan sea algo peor que el empleo de un índice para $númd$, pero es una alternativa razonable. Por tanto, si ya se tiene un índice para $edad$ (debido a alguna otra consulta de la carga de trabajo), esta variante de la consulta de ejemplo no justifica la creación de un índice para el campo $númd$ de Empleados.

La siguiente consulta implica una selección por rango:

```

SELECT E.nombree, D.nombred
FROM   Empleados E, Departamentos D
    
```

¹Ésta es tan sólo una aproximación heurística. Si $nombred$ no es la clave, y no se tienen estadísticas para comprobar esta suposición, es posible que varias tuplas satisfagan la condición.

```

WHERE E.sueldo BETWEEN 10000 AND 20000
      AND E.afición='Sellos' AND E.númd=D.númd

```

Esta consulta ilustra el empleo del operador `BETWEEN` para la expresión de selecciones por rangos. Es equivalente a la condición:

$$10000 \leq E.sueldo \text{ AND } E.sueldo \leq 20000$$

Se recomienda el empleo de `BETWEEN` para expresar condiciones por rangos; facilita tanto al usuario como al optimizador el reconocimiento de las dos partes de la selección por rango.

Volviendo a la consulta de ejemplo, las dos selecciones (que no son reuniones) se realizan sobre la relación Empleados. Por tanto, está claro que lo mejor es un plan en el que Empleados sea la relación externa y Departamentos la interna, como en la consulta anterior, y habría que construir un índice asociativo para el atributo *númd* de Departamentos. ¿Pero qué índice habría que construir para Empleados? Un índice de árbol B+ para el atributo *sueldo* sería útil para la selección por rango, especialmente si está agrupado. Un índice asociativo para el atributo *afición* sería útil para la selección de igualdad. Si se dispone de uno de estos índices se pueden recuperar tuplas de Empleados utilizando el índice para *númd* y aplicar todas las demás selecciones y proyecciones sobre la marcha. Si se dispone de los dos índices, el optimizador escogerá el más selectivo para cada consulta; es decir, considerará qué selección (la condición de rango para *sueldo* o la igualdad para *afición*) tiene menos tuplas que la cumplan. En general, la selectividad de los índices depende de los datos. Si hay muy pocas personas con sueldos en el rango dado y muchas coleccionan sellos, es mejor el índice de árbol B+. En caso contrario, es mejor el índice asociativo para *afición*.

Si se conocen las constantes de la consulta (como en este ejemplo), se pueden estimar las diferentes selectividades si se dispone de estadísticas de los datos. En caso contrario, como regla general, es probable que las selecciones de igualdad sean más selectivas, y sería una decisión razonable crear un índice asociativo para *afición*. A veces no se conocen las constantes de las consultas —puede que se obtenga una consulta mediante la ampliación de otra realizada a una vista en el momento de la ejecución o que se tenga una consulta en SQL dinámico, que permite que las constantes se especifiquen como *variables comodín* (por ejemplo, `%X`) y se asignen en el momento de la ejecución (véanse los Apartados 6.1.3 y 6.1.2)—. En ese caso, si la consulta es muy importante, puede que se decida crear un índice de árbol B+ para *sueldo* y un índice asociativo para *afición* y se deje que la elección la haga el optimizador en el momento de la ejecución.

13.4 AGRUPACIÓN E INDEXACIÓN

Los índices agrupados pueden resultar de especial importancia para el acceso a la relación interna de una reunión de índices de bucles anidados. Para comprender la reunión entre los índices agrupados y las reuniones se retomará el primer ejemplo:

```

SELECT E.nombree, D.director
FROM   Empleados E, Departamentos D
WHERE  D.nombred='Juguetes' AND E.númd=D.númd

```

Se ha concluido que emplear un índice para *nombred* para recuperar las tuplas de Departamentos que satisfacen la condición para *nombred* y hallar las tuplas coincidentes de Empleados mediante un índice para *númd* es un buen plan de evaluación. ¿Deben estar agrupados estos índices? Dada la suposición de que es probable que el número de tuplas que satisfacen *D.nombred*=‘Juguetes’ sea pequeño, se debe construir un índice no agrupado para *nombred*. Por otro lado, Empleados es la relación interna de una reunión de índices de bucles anidados y *númd* no es una clave candidata. Esta situación es un poderoso argumento a favor de la agrupación del índice para el campo *númd* de Empleados. De hecho, dado que la reunión consiste en plantear repetidamente selecciones de igualdad para el campo *númd* de la relación interna, este tipo de consulta es una justificación más potente para la agrupación del índice para *númd* que una mera consulta de selección como la selección anterior para *afición*. (Por supuesto, también hay que tener en cuenta factores como la selectividad y la frecuencia de las consultas.)

El ejemplo siguiente, muy parecido al anterior, ilustra la manera en que se pueden utilizar los índices agrupados para las reuniones de ordenación-mezcla:

```
SELECT E.nombree, D.director
  FROM Empleados E, Departamentos D
 WHERE E.afición='Sellos' AND E.númd=D.númd
```

Esta consulta se diferencia de la anterior en que la condición *E.afición*=‘Sellos’ sustituye a *D.nombred*=‘Juguetes’. Con base en la suposición de que hay pocos empleados en el departamento Juguetes, se han escogido índices que faciliten una reunión indexada de bucles anidados con Departamentos como relación externa. Supóngase ahora que muchos empleados coleccionan sellos. En ese caso, puede que fuera más eficiente una reunión por bloques de bucles anidados, o por ordenación-mezcla. Las reuniones por ordenación-mezcla pueden aprovechar el índice de árbol B+ agrupado por el atributo *númd* de Departamentos para recuperar tuplas y, por tanto, evitar ordenar Departamentos. Obsérvese que no resultan útiles los índices no agrupados —ya que se recuperan todas las tuplas, es probable que realizar una operación de E/S por cada tupla resulte prohibitivamente costoso—. Si no hay ningún índice para el campo *númd* de Empleados, se pueden recuperar las tuplas de Empleados (posiblemente empleando un índice para *afición*, especialmente si el índice está agrupado), aplicar la selección *E.afición*=‘Sellos’ sobre la marcha y ordenar las tuplas que cumplen las condiciones según el valor de *númd*.

Como ha indicado este análisis, cuando se recuperan tuplas mediante un índice, las consecuencias del agrupamiento dependen del número de tuplas recuperadas, es decir, del número de tuplas que satisfacen las condiciones de selección que coinciden con el índice. Los índices no agrupados son igual de buenos que los agrupados para las selecciones que sólo recuperan una tupla (por ejemplo, las selecciones de igualdad sobre claves candidatas). A medida que aumenta el número de tuplas recuperadas, el índice no agrupado se vuelve rápidamente más costoso incluso que una exploración secuencial de toda la relación. Aunque la exploración secuencial recupera todas las tuplas, cada página sólo se recupera una vez, mientras que, si se emplea un índice no agrupado, se puede recuperar cada página tantas veces como tuplas contenga. Si se llevan a cabo operaciones de E/S bloqueadas (como suele ocurrir) es aún mayor la ventaja relativa de la exploración secuencial respecto del índice no agrupado. (Por

supuesto, las operaciones de E/S bloqueadas aceleran también el acceso cuando se emplean índices agrupados.)

La relación entre el número de tuplas recuperadas, vistas como porcentaje del número total de tuplas de la relación, y el coste de los diferentes métodos de acceso se ilustra en la Figura 13.2. Por simplificar, se da por supuesto que la consulta es una selección sobre una única relación. (Obsérvese que esta figura refleja el coste de escritura del resultado; en otro caso, la línea de la exploración secuencial sería plana.)

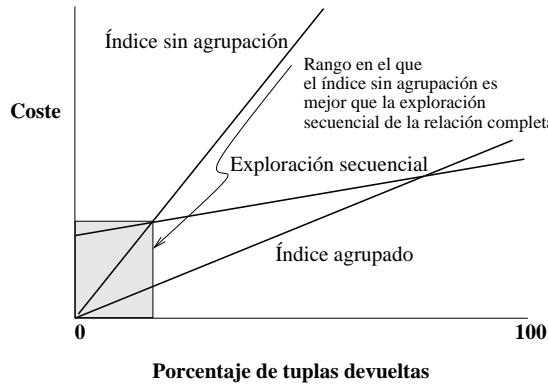


Figura 13.2 Consecuencias del agrupamiento

13.4.1 Coagrupación de dos relaciones

Aunque cada archivo suele contener sólo los registros de una de las relaciones, algunos sistemas permiten que se guarden los registros de más de una relación en un mismo archivo. El usuario de bases de datos puede solicitar que los registros de dos relaciones se intercalen físicamente de esta manera. Esta disposición de los datos se conoce a veces como **coagrupación** de las dos relaciones. A continuación se analizará cuándo puede resultar beneficiosa la coagrupación.

Por ejemplo, considérense dos relaciones con los esquemas siguientes:

```
Repuestos(idr: integer, nombre: string, coste: integer, idproveedor: integer)
Montaje(idrepuesto: integer, idcomponente: integer, cantidad: integer)
```

En este esquema se pretende que el campo *idcomponente* de Montaje sea el *idr* de algún repuesto que se emplea como componente para montar el repuesto con *idr* igual a *idrepuesto*. Por tanto, la tabla Montaje representa una relación 1:N entre los repuestos y sus componentes; cada repuesto puede tener muchos componentes pero, a lo sumo, puede ser componente de un único repuesto. En la tabla Repuestos, *idr* es la clave. Para los repuestos compuestos (los montados a partir de otros repuestos, como indica el contenido de Montaje), el campo *coste* es igual al coste de montar ese repuesto a partir de sus componentes.

Supóngase que una consulta frecuente es averiguar los componentes (inmediatos) de todos los repuestos de un proveedor dado:

```
SELECT R.idr, M.idcomponente
```

```
FROM Repuestos R, Montaje M
WHERE R.idr = M.idrepuesto AND R.idproveedor = 'Acme'
```

Un buen plan de evaluación es aplicar la condición de selección a Repuestos y recuperar luego las tuplas coincidentes de Montaje mediante un índice para el campo *idrepuesto*. Lo ideal sería que el índice para *idrepuesto* estuviera agrupado. Este plan es razonablemente bueno. Sin embargo, si estas selecciones son frecuentes y se desea optimizarlas más, se pueden *coagrupar* las dos tablas. En este enfoque se guardan juntos los registros de las dos tablas, con cada registro *R* de Repuestos seguido de todos los registros *M* de montajes tales que *R.idr = M.idrepuesto*. Este enfoque es mejor que almacenar por separado las dos relaciones y tener un índice agrupado para *idrepuesto*, ya que no necesita buscar por el índice para hallar los registros de Montaje que coinciden con cada registro de Repuestos. Por tanto, para cada consulta de selección se ahorran unas cuantas (generalmente dos o tres) operaciones de E/S de las páginas del índice.

Si se está interesado en averiguar los componentes inmediatos de *todos* los repuestos (es decir, la consulta anterior sin selección sobre *idproveedor*), la creación de un índice agrupado para *idrepuesto* y la realización de una reunión de bucles anidados de índices con Montaje como relación interna ofrece un buen rendimiento. Una estrategia aún mejor es crear un índice agrupado para el campo *idrepuesto* de Montaje y el campo *idr* de Repuestos, realizar luego una reunión de ordenación-mezcla, empleando los índices para recuperar las tuplas de manera ordenada. Esta estrategia es comparable a realizar la reunión empleando una organización coagrupada, que implica sólo una exploración del conjunto de tuplas (de Repuestos y de Montaje, que se guardan juntas de modo intercalado).

La verdadera ventaja de la coagrupación se ilustra en la consulta siguiente:

```
SELECT R.idr, M.idcomponente
FROM Repuestos R, Montaje M
WHERE R.idr = M.idrepuesto AND R.coste=10
```

Supóngase que muchos repuestos tienen *coste* = 10. Esta consulta equivale esencialmente a un conjunto de consultas en el que se nos da un registro de Repuestos y se desea hallar registros coincidentes de Montaje. Si se tiene un índice del campo *coste* de Repuestos, se pueden recuperar las tuplas de Repuestos que cumplen las condiciones fijadas. Para cada una de esas tuplas hay que emplear el índice de Montaje para localizar los registros con el *idr* dado. Se evita el acceso mediante índice a Montaje si se tiene una organización coagrupada. (Por supuesto, sigue haciendo falta un índice para el atributo *coste* de las tuplas de Repuestos.)

Una optimización así resulta especialmente importante si se desea atravesar varios niveles de la jerarquía repuesto-componente. Por ejemplo, una consulta frecuente es la búsqueda del coste total de un repuesto, que necesita que se lleven a cabo repetidamente reuniones de Repuestos y de Montaje. Por cierto, si no se conoce previamente el número de niveles de la jerarquía, el número de reuniones varía y la consulta no se puede expresar en SQL. La consulta se puede responder incluyendo una instrucción de SQL para la reunión en un programa iterativo del lenguaje anfitrión. La manera de expresar la consulta es ortogonal a nuestro principal objetivo actual, que es que la coagrupación resulta especialmente ventajosa si la reunión en cuestión se lleva a cabo muy frecuentemente (bien porque surge repetidamente en una consulta importante como la búsqueda del coste total, bien porque la propia consulta de reunión se formula frecuentemente).

Para resumir la coagrupación:

- Puede acelerar las reuniones, en especial las reuniones clave-clave externa correspondientes a relaciones 1:N.
- La exploración secuencial de cualquiera de las relaciones se hace más lenta. (En el ejemplo anterior, ya que las tuplas de Montaje se guardan entre las tuplas consecutivas de Repuestos, la exploración de todas las tuplas de Repuestos se hace más lenta que si las tuplas de Repuestos estuvieran guardadas por separado. De manera parecida, la exploración secuencial de todas las tuplas de Montaje también es más lenta.)
- Todas las inserciones, borrados y actualizaciones que modifican la longitud de los registros se vuelven más lentas, debido a las sobrecargas motivadas por el mantenimiento de la agrupación. (No se examinarán aquí los problemas de implementación relacionados con la coagrupación.)

13.5 ÍNDICES QUE PERMITEN PLANES EXCLUSIVAMENTE PARA ÍNDICES

Este apartado toma en consideración varias consultas para las cuales se pueden hallar planes eficientes que evitan la recuperación de las tuplas de una de las relaciones a las que se hace referencia; en su lugar, estos planes exploran un índice asociado (que es probable que sea mucho más pequeño). Los índices que se utilizan (sólo) para las exploraciones exclusivas de índices tienen que estar agrupados, ya que no se recuperan las tuplas de la relación indexada.

Esta consulta recupera los directores de los departamentos que tienen, como mínimo, un empleado:

```
SELECT D.director
  FROM Departamentos D, Empleados E
 WHERE D.númd=E.númd
```

Obsérvese que no se conserva ningún atributo de Empleados. Si se tiene un índice para el campo *númd* de Empleados, se puede aplicar la optimización de la reunión de bucles anidados de índices que emplea una exploración exclusiva de índices para la relación interna. Dada esta variante de la consulta, la decisión correcta es crear un índice no agrupado para el campo *númd* de Empleados, en vez de un índice agrupado.

La consulta siguiente lleva esta idea un poco más lejos:

```
SELECT D.director, E.ide
  FROM Departamentos D, Empleados E
 WHERE D.númd=E.númd
```

Si se tiene un índice para el campo *númd* de Empleados, se puede utilizar para recuperar tuplas de Empleados durante la reunión (con Departamentos como relación externa) pero, a menos que el índice esté agrupado, este enfoque no será eficiente. Por otro lado, supóngase que se tiene un índice de árbol B+ para $\langle \text{númd}, \text{ide} \rangle$. Ahora toda la información necesaria sobre las tuplas de Empleados está incluida en la entrada de datos de esa tupla en el índice.

Se puede emplear el índice para averiguar la primera entrada de datos con un *númd* dado; todas las entradas de datos con el mismo *númd* se guardan juntas en el índice. (Obsérvese que no se puede utilizar un índice asociativo para la clave compuesta $\langle \text{númd}, \text{ide} \rangle$ para localizar una entrada sólo con un *númd* dado.) Por tanto, se puede evaluar esta consulta mediante una reunión de bucles anidados de índices con Departamentos como relación externa y una exploración exclusiva de índices de la relación interna.

13.6 HERRAMIENTAS PARA LA SELECCIÓN DE ÍNDICES

El número de índices posibles a crear puede llegar a ser muy grande: para cada relación se pueden llegar a considerar todos los subconjuntos de atributos posibles como claves del índice; hay que decidir sobre la ordenación de los atributos en el índice; y también hay que decidir los índices que se deben agrupar y los que se deben dejar sin agrupar. Muchas aplicaciones de gran tamaño —por ejemplo, los sistemas de planificación de recursos de las empresas— crean decenas de millares de relaciones diferentes, y el ajuste manual de esquemas tan grandes es un empeño desalentador.

La dificultad y la importancia de la tarea de selección de los índices provocó el desarrollo de herramientas que ayudan a los administradores de bases de datos a seleccionar índices adecuados para cada carga de trabajo. La primera generación de estos **asistentes para el ajuste de los índices**, o **asesores sobre índices**, estaba formada por herramientas independientes, ajenas al motor de la base de datos; sugerían índices a crear dada la carga de trabajo de las consultas de SQL. El principal inconveniente de estos sistemas era que tenían que replicar el modelo de costes del optimizador de consultas de la base de datos en la herramienta de ajuste para garantizar que el optimizador escogiera los mismos planes de evaluación de las consultas que la herramienta de diseño. Dado que los optimizadores de consultas cambian de una versión a otra de los sistemas comerciales de bases de datos, hacía falta un esfuerzo considerable para mantener sincronizados la herramienta de ajuste y el optimizador de la base de datos. La generación más reciente de herramientas de ajuste se integra con el motor de la base de datos y emplea el optimizador de consultas de la base de datos para estimar el coste de la carga de trabajo dado un conjunto de índices, lo que evita la duplicación del modelo de costes del optimizador de consultas en una herramienta externa.

13.6.1 Selección automática de índices

El conjunto de índices para un esquema dado de la base de datos se denomina **configuración de índices**. Se da por supuesto que la carga de trabajo de cada consulta es un conjunto de consultas sobre el esquema de una base de datos en el que cada consulta tiene asignada una frecuencia de aparición. Dados el esquema de una base de datos y una carga de trabajo, el **coste de la configuración de índices** es el coste esperado de la ejecución de las consultas de la carga de trabajo dada la configuración de índices —teniendo en cuenta las diferentes frecuencias de las consultas de la carga de trabajo—. Dado el esquema de una base de datos y la carga de trabajo de una consulta ya se puede definir el problema de la **selección automática de índices** como la búsqueda de una configuración de índices con coste mínimo.

Al igual que ocurre con la optimización de consultas, en la práctica el objetivo es hallar una *buen*a configuración de índices más que la auténtica configuración óptima.

¿Por qué la selección automática de índices es un problema difícil? Calculemos el número de índices diferentes con c atributos, suponiendo que la tabla tiene n atributos. Para el primer atributo del índice hay n posibilidades, para el segundo atributo, $n - 1$ y, por tanto, para un índice con c atributos hay un total de $n \cdot (n - 1) \cdots (n - c + 1) = \frac{n!}{(n - c)!}$ posibles índices diferentes. El número total de índices diferentes con un máximo de c es:

$$\sum_{i=1}^c \frac{n!}{(n - i)!}$$

Para una tabla con 10 atributos hay diez índices diferentes con un solo atributo, 90 con dos atributos y 30.240 con cinco atributos. Para una carga de trabajo compleja que implique a centenares de tablas el número de configuraciones de índices posibles, evidentemente, es muy grande.

La eficiencia de las herramientas para la selección automática de índices puede dividirse en dos componentes: (1) el número de posibles configuraciones de índices consideradas y (2) el número de llamadas al optimizador necesarias para evaluar el coste de cada configuración. Obsérvese que reducir el espacio de búsqueda de los posibles índices es análogo a restringir el espacio de búsqueda del optimizador de consultas a los planes profundos por la izquierda. En muchos casos el plan óptimo no es profundo por la izquierda, pero entre todos los planes profundos por la izquierda suele haber un plan cuyo coste es cercano al del plan óptimo.

Se puede disminuir fácilmente el tiempo empleado en la selección automática de índices reduciendo el número de posibles configuraciones de índices, pero cuanto menor sea el espacio de configuraciones de índices considerado, más lejos se hallará la configuración de índices definitiva de la óptima. Por tanto, los diferentes asistentes para el ajuste de índices recortan de manera diferente el espacio de búsqueda, por ejemplo, considerando sólo índices con uno o con dos atributos.

13.6.2 Funcionamiento de los asistentes para el ajuste de índices

Todos los asistentes para el ajuste de índices examinan el conjunto de índices posibles para encontrar la configuración de índices de coste mínimo. Las diversas herramientas se diferencian en el espacio de posibles configuraciones de índices que toman en consideración y en el modo en que examinan ese espacio. Se describirá un algoritmo representativo; las herramientas existentes implementan variantes de este algoritmo, pero todas las implementaciones tienen la misma estructura básica.

Antes de describir el algoritmo de ajuste de índices, considérese el problema de estimar el coste de cada configuración. Obsérvese que no es factible crear realmente el conjunto de índices de la posible configuración y optimizar luego la carga de trabajo de consultas dada la configuración física de los índices. Incluso la creación de una sola posible configuración con varios índices tardaría horas en bases de datos de gran tamaño y representaría una carga considerable para el propio sistema de bases de datos. Dado que se desea examinar un gran número de configuraciones posibles, este enfoque no resulta factible.

El Asesor de índices de DB2. El Asesor de índices de DB2 es una herramienta para la recomendación automática de índices dada una carga de trabajo. La carga de trabajo se guarda en el sistema de bases de datos en una tabla denominada ADVISE_WORKLOAD. Contiene (1) instrucciones de SQL de la caché dinámica de instrucciones de SQL de DB2, una caché para las instrucciones de SQL ejecutadas recientemente (2) instrucciones de SQL procedentes de paquetes —grupos de instrucciones de SQL compiladas de manera estática— o (3) instrucciones de SQL de un controlador en línea denominado Patrullero de Consultas. El Asesor de DB2 permite que el usuario especifique la cantidad máxima de espacio de disco que pueden ocupar los nuevos índices y el tiempo máximo para el cálculo de la configuración de índices recomendada.

El Asesor de índices de DB2 consiste en un programa que examina de manera inteligente un subconjunto de configuraciones de índices. Dada una posible configuración, llama al optimizador de consultas para cada consulta de la tabla ADVISE_WORKLOAD primero en el modo RECOMMEND_INDEXES, en el que el optimizador recomienda un conjunto de índices y los guarda en la tabla ADVISE_INDEXES. En el modo EVALUATE_INDEXES el optimizador evalúa las ventajas de la configuración de índices para cada consulta de la tabla ADVISE_WORKLOAD. El resultado del paso de ajuste de los índices son instrucciones del LDD de SQL cuya ejecución crea los índices recomendados.

El Asistente para ajuste de índices de Microsoft SQL Server 2000. Microsoft fue el primero en implementar un asistente para ajuste integrado con el optimizador de consultas de la base de datos. El Asistente para ajuste de Microsoft tiene tres modos de ajuste que permiten al usuario equilibrar el tiempo de ejecución del análisis con el número de posibles configuraciones de índices examinadas: *rápido*, *medio* y *completo*, de los que el modo *rápido* rápido tiene el menor tiempo de ejecución y el modo *completo* examina el mayor número de configuraciones. Para reducir aún más el tiempo de ejecución, la herramienta tiene un modo de muestreo en el que el asistente para ajuste muestrea aleatoriamente las consultas de la carga de trabajo entrante para acelerar el análisis. Otros parámetros son el espacio máximo permitido para los índices recomendados, el número máximo de atributos por cada índice considerado y las tablas para las que se pueden generar índices. El Asistente para ajuste de índices de Microsoft también permite el *escalado de tablas*, en el que el usuario puede especificar el número de registros esperado para las tablas implicadas en la carga de trabajo. Esto permite que los usuarios planifiquen el crecimiento futuro de las tablas.

Por tanto, los algoritmos de ajuste de índices suelen *simular* el efecto de los índices en la posible configuración (a menos que tales índices existan ya). Esos índices **hipotéticos** son para el optimizador de consultas o cualquier otro índice y se tienen en cuenta a la hora de calcular el coste de la carga de trabajo para una configuración dada, pero su creación no supone la sobrecarga de la creación real de índices. Los sistemas comerciales de bases de datos que ofrecen asistentes para el ajuste de índices que emplean el optimizador de consultas de la base de datos se han ampliado con un módulo que permite la creación y eliminación de índices hipotéticos con las estadísticas necesarias de esos índices (que se emplean al estimar el coste de los planes de consultas).

A continuación se examinará un algoritmo de ajuste de índices representativo. El algoritmo procede en dos fases, la *selección de posibles índices* y la *enumeración de configuraciones*. En la primera se selecciona un conjunto de posibles índices que se tomarán en consideración durante la segunda fase como elementos para la creación de configuraciones de índices. A continuación se examinarán con más detalle estas dos fases.

Selección de posibles índices

En el apartado anterior se vio que es imposible tomar en consideración todos los índices posibles debido al enorme número de posibles índices disponibles para los esquemas de bases de datos de mayor tamaño. Una heurística para recortar el gran espacio de índices posibles es ajustar en primer lugar todas las consultas de la carga de trabajo de manera independiente y seleccionar luego la unión de los índices seleccionados en esta primera etapa como datos de entrada para la segunda fase.

Para las consultas se introducirá ahora el concepto de atributo indexable, que es un atributo cuya aparición en un índice puede cambiar el coste de la consulta. Un **atributo indexable** es un atributo para el que la parte **WHERE** de la consulta tiene una condición (por ejemplo, un predicado de igualdad), o bien ese atributo aparece en una cláusula **GROUP BY** o **ORDER BY** de la consulta SQL. Un **índice admisible** para una consulta es un índice que sólo contiene atributos indexables para esa consulta.

¿Cómo seleccionar los posibles índices para una consulta concreta? Un enfoque es la mera enumeración de todos los índices con un máximo de k atributos. Se empieza con todos los atributos indexables como posibles índices con un solo atributo, luego se añaden todas las combinaciones de dos atributos indexables y se repite este procedimiento hasta el umbral de tamaño k definido por el usuario. Evidentemente, este procedimiento resulta muy costoso, ya que se añaden $n + n \cdot (n - 1) + \dots + n \cdot (n - 1) \dots (n - k + 1)$ posibles índices, pero garantiza que el mejor índice con un máximo de k atributos esté entre los índices posibles. Las referencias al final de este capítulo contienen indicaciones de algoritmos heurísticos de búsqueda más rápidos (pero menos exhaustivos).

Enumeración de configuraciones

En la segunda fase se emplean los posibles índices para enumerar las configuraciones de índices. Al igual que en la primera fase, se pueden enumerar de manera exhaustiva todas las configuraciones de índices con un tamaño máximo de k , esta vez combinando posibles índices.

Como en la fase anterior, son posibles estrategias más sofisticadas que recorten el número de configuraciones tomadas en consideración sin dejar de generar una configuración final de calidad elevada (es decir, coste de ejecución reducido para la carga de trabajo final).

13.7 VISIÓN GENERAL DEL AJUSTE DE BASES DE DATOS

Tras la fase inicial de diseño de la base de datos, su uso real ofrece una valiosa fuente de información detallada que se puede utilizar para refinar el diseño inicial. Muchas de las suposiciones originales sobre la carga de trabajo esperada se pueden sustituir por patrones de uso observados; en general, parte de la especificación inicial de la carga de trabajo se valida y parte resulta ser errónea. Las conjeturas iniciales sobre el tamaño de los datos se pueden sustituir por estadísticas reales obtenidas de los catálogos del sistema (aunque esta información no deja de modificarse a medida que el sistema evoluciona). El control meticuloso de las consultas puede revelar problemas inesperados; por ejemplo, puede que el optimizador no emplee como se pretendía algunos índices para la generación de buenos planes.

El ajuste continuo de la base de datos es importante para obtener el mejor rendimiento posible. En este apartado se presentarán tres tipos de ajuste: *ajuste de índices*, *ajuste del esquema conceptual* y *ajuste de consultas*. El análisis de la selección de índices es aplicable también a las decisiones relativas al ajuste de índices. El esquema conceptual y el ajuste de consultas se examinan más a fondo en los Apartados 13.8 y 13.9.

13.7.1 Ajuste de índices

La selección de índices inicial se puede refinar por varias razones. La más sencilla es que la carga de trabajo observada revele que algunas consultas y actualizaciones consideradas importantes en la especificación de la carga de trabajo inicial no sean muy frecuentes. Puede que la carga de trabajo observada también identifique consultas y actualizaciones nuevas que *sean* importantes. Hay que revisar la elección inicial de índices a la vista de esta nueva información. Se puede descartar parte de los índices originales y añadir otros nuevos. El razonamiento subyacente es parecido al empleado en el diseño inicial.

Puede que también se descubra que el optimizador de un sistema dado no encuentra parte de los planes que se esperaba que encontrara. Por ejemplo, considérese la consulta siguiente, que ya se ha analizado anteriormente:

```
SELECT D.director
  FROM Empleados E, Departamentos D
 WHERE D.nombred='Juguetes' AND E.númd=D.númd
```

En este caso sería un buen plan emplear un índice para *nombred* para recuperar las tuplas de Departamentos con *nombred='Juguetes'* y emplear un índice para el campo *númd* de Empleados como relación interna, empleando una exploración exclusiva de índices. Anticipando que el optimizador encontrará un plan así, se podría haber creado un índice no agrupado para el campo *númd* de Empleados.

Supóngase ahora que las consultas de esta forma tardan un tiempo inesperadamente largo en ejecutarse. Se puede solicitar ver el plan generado por el optimizador. (La mayor parte de los sistemas comerciales ofrecen una orden sencilla para hacerlo.) Si el plan indica que no se utiliza una exploración exclusiva de índices, pero que se están recuperando tuplas de Empleados, hay que volver a considerar la elección inicial de índices, dada esta revelación sobre las (desgraciadas) limitaciones del sistema. Una alternativa que se puede tomar en consideración en este caso sería descartar el índice no agrupado para el campo *númd* de Empleados y sustituirlo por un índice agrupado.

Otras limitaciones frecuentes de los optimizadores es que no tratan las selecciones que implican expresiones de cadenas de caracteres, aritméticas o valores *null* de manera efectiva. Estos puntos se examinarán más a fondo cuando se considere el ajuste de consultas en el Apartado 13.9.

Además de volver a examinar la elección de índices, es conveniente reorganizar algunos índices de manera periódica. Por ejemplo, puede que un índice estático, como los índices ISAM, haya desarrollado largas cadenas de desbordamiento. Descartar ese índice y reconstruirlo —si es factible, dado el acceso interrumpido a la relación indexada— puede mejorar de forma sustancial los tiempos de acceso cuando se emplea ese índice. Incluso para estructuras dinámicas como los árboles B+, si la implementación no mezcla las páginas para las eliminaciones, la ocupación del espacio puede disminuir considerablemente en algunas situaciones. Esto, a su vez, hace que el tamaño del índice (medido en páginas) sea mayor de lo necesario, y podría aumentar la altura y, por tanto, el tiempo de acceso. Se debería considerar la reconstrucción del índice. Las actualizaciones generalizadas de índices agrupados también pueden llevar a la asignación de páginas de desbordamiento, lo que disminuye el nivel de agrupación. Una vez más, puede que merezca la pena reconstruir el índice.

Finalmente, téngase en cuenta que el optimizador de consultas confía en las estadísticas generadas en los catálogos del sistema. Estas estadísticas sólo se actualizan cuando se ejecuta un programa de utilidad especial; hay que asegurarse de ejecutar esa utilidad con la suficiente frecuencia como para mantener razonablemente al día las estadísticas.

13.7.2 Ajuste del esquema conceptual

En el transcurso del diseño de la base de datos puede que se haga evidente que la vigente elección de esquemas de relación no permite cumplir los objetivos de rendimiento para la carga e trabajo dada con ningún conjunto (factible) de opciones de diseño físico. En ese caso, puede que haya que rediseñar el esquema conceptual (y reexaminar las decisiones sobre el diseño físico afectadas por las modificaciones que se hagan).

Puede que se haga necesario un cambio de diseño durante el proceso inicial de diseño o más tarde, una vez el sistema lleva algún tiempo en uso. Una vez diseñada la base de datos y rellena de tuplas, la modificación del esquema conceptual exige un significativo esfuerzo en términos de asignación del contenido de las relaciones afectadas. Pese a todo, puede que sea necesario revisar el esquema conceptual a la vista de nuestra experiencia con el sistema. (Estos cambios en el esquema de un sistema operativo se denominan a veces **evoluciones del esquema**.) A continuación se considerarán los aspectos relativos al (re)diseño del esquema conceptual desde el punto de vista del rendimiento.

Lo primero que hay que entender es que *la elección del esquema conceptual debe venir orientada por la toma en consideración de las consultas y actualizaciones de la carga de trabajo*, además de los aspectos de redundancia que motivan las normalización (que se analizaron en el Capítulo 12). Se deben tomar en consideración varias opciones a la hora de ajustar el esquema conceptual:

- Puede que se decida optar por un diseño en 3FN en lugar de uno en FNBC.
- Si hay dos maneras de descomponer un esquema dado en 3FN o en FNBC, la opción debería estar guiada por la carga de trabajo.
- Puede que a veces se decida descomponer aún más una relación que *ya* se encuentra en FNBC.
- En otras situaciones puede que se *desnormalice*. Es decir, puede que se decida sustituir un conjunto de relaciones obtenidas mediante la descomposición de una relación mayor por la relación original (de mayor tamaño), aunque sufra problemas de redundancia. De manera alternativa, puede que se decida añadir algunos campos a ciertas relaciones para acelerar algunas consultas importantes, aunque esto provoque el almacenamiento redundante de parte de la información (y, en consecuencia, un esquema que no se halle ni en 3FN ni en FNBC).
- Este análisis de la normalización se ha concentrado en la técnica de *descomposición*, que equivale a la partición vertical de las relaciones. Otra técnica que se debe considerar es la *partición horizontal* de las relaciones, que llevaría a tener dos relaciones con esquemas idénticos. Obsérvese que no se habla de partir físicamente las tuplas de una sola relación; en cambio, se desea crear dos relaciones diferentes (posiblemente con restricciones e índices diferentes para cada una de ellas).

Por otro lado, cuando se rediseña el esquema conceptual, especialmente si se está ajustando el esquema de una base de datos ya existente, merece la pena tomar en consideración si se deben crear vistas que oculten estos cambios a la vista de los usuarios para los que el esquema original resulte más natural. Las opciones relativas al ajuste del esquema conceptual se analizan en el Apartado 13.8.

13.7.3 Ajuste de consultas y de vistas

Si se observa que una consulta se ejecuta mucho más lentamente de lo esperado, hay que examinar la consulta con detenimiento para localizar el problema. Una pequeña reescritura de la consulta, quizás junto con algún ajuste de índices, suele bastar para solucionar el problema. Un ajuste parecido se puede realizar si las consultas a alguna vista se ejecutan más lentamente de lo esperado. No se examinará aquí por separado el ajuste de las vistas; basta con considerar las consultas a vistas como consultas propiamente dichas (después de todo, las consultas a vistas se amplian para tener en cuenta la definición de la vista antes de optimizarlas) y pensar cómo ajustarlas.

Al ajustar una consulta, lo primero que hay que comprobar es que el sistema emplea el plan que se espera que emplee. Quizás el sistema no halla el mejor plan por diferentes razones.

Algunas situaciones frecuentes que muchos optimizadores no manejan eficientemente son las siguientes:

- Una condición de selección que implica valores *null*.
- Condiciones de selección que implican expresiones aritméticas o de cadenas de caracteres o condiciones que emplean la conectiva OR. Por ejemplo, si se tiene la condición $E.edad = 2*D.edad$ en la cláusula WHERE, puede que el optimizador utilice correctamente un índice disponible para *E.edad* pero no logre utilizar un índice disponible para *D.edad*. La sustitución de la condición por $E.edad/2 = D.edad$ corregiría el problema.
- Incapacidad para reconocer planes sofisticados, como una exploración exclusiva de índices para una consulta de agregación que implique una cláusula GROUP BY. Por supuesto, prácticamente ningún optimizador busca planes fuera del espacio de planes habitual, como los árboles de reunión que no es profunda por la izquierda. Por tanto, es importante una buena comprensión de lo que hacen normalmente los optimizadores. Además, cuanto más se conozca de las ventajas e inconvenientes de un sistema dado, mejor preparado se estará.

Si el optimizador no es tan inteligente como para hallar el mejor plan (empleando los métodos de acceso y las estrategias de evaluación soportados por el SGBD), algunos sistemas permiten a los usuarios dirigir la elección de plan facilitando sugerencias al optimizador; por ejemplo, puede que los usuarios logren obligar al empleo de un índice concreto o elijan el orden y el método de las reuniones. Los usuarios que deseen orientar de esta manera la optimización deben tener un profundo conocimiento de la optimización y de las posibilidades del SGBD dado. El ajuste de consultas se analiza con más detenimiento en el Apartado 13.9.

13.8 OPCIONES DE AJUSTE DEL ESQUEMA CONCEPTUAL

A continuación se mostrarán las opciones de ajuste del esquema conceptual mediante varios ejemplos que utilizan los esquemas siguientes:

```
Contratos(idc: integer, idproveedor: integer, idprojeto: integer,
          iddep: integer, idrepuesto: integer, cantidad: integer, valor: real)
Departamentos(idd: integer, presupuesto: real, informemanual: varchar)
Repuestos(idr: integer, coste: integer)
Proyectos(idpr: integer, dir: char(20))
Proveedores(idp: integer, direccion: char(50))
```

En aras de la brevedad se suele emplear el convenio habitual de denotar los atributos mediante un solo carácter y los esquemas de las relaciones por una secuencia de caracteres. Considérese el esquema para la relación Contratos, que se denota como IPYDRCV, donde cada letra denota un atributo. El significado de cada tupla de esta relación es que el contrato con *idc* C es un acuerdo para que el proveedor P (con *idp* igual a *idproveedor*) suministre C unidades del repuesto R (con *idr* igual a *idrepuesto*) al proyecto Y (con *idpr* igual a *idprojeto*)

asociado al departamento D (con $iddep$ igual a idd), y que el valor V de este contrato es igual a $valor^2$.

Hay dos restricciones de integridad conocidas con respecto a Contratos. Los proyectos adquieren cada repuesto mediante un solo contrato; por tanto, no puede haber dos contratos diferentes en los que el mismo proyecto compre el mismo repuesto. Esta restricción se representa mediante la DF $YR \rightarrow I$. Además, cada departamento compra, como máximo, un repuesto a cada proveedor. Esta restricción se representa mediante la DF $PD \rightarrow R$. Además, por supuesto, el identificador de los contratos, I, es una clave. El significado de las otras relaciones debería ser evidente y no se describirán más a fondo porque nos centraremos en la relación Contratos.

13.8.1 Aceptación de formas normales más débiles

Considérese la relación Contratos. ¿Se debe descomponer en relaciones de menor tamaño? Veamos en qué forma normal se encuentra. Las posibles claves de esta relación son I e YR. (I se da como clave, e YR determina funcionalmente a I.) La única dependencia que no afecta a clave alguna es $PD \rightarrow R$, y R es un atributo de *primera clase*, ya que es parte de la posible clave YR. Por tanto, la relación no se halla en FNBC —puesto que hay una dependencia que no afecta a clave alguna— pero sí en 3FN.

Al emplear la dependencia $PD \rightarrow R$ para orientar la descomposición se obtienen los dos esquemas PDR e IPYDCV. Esta descomposición no tiene pérdida, pero no conserva las dependencias. Sin embargo, al añadir el esquema de relación IYR se obtienen una descomposición por reunión sin pérdida en FNBC que conserva la dependencia. Empleando la directriz que indica que esta descomposición en FNBC es adecuada, se puede decidir sustituir Contratos por tres relaciones con los esquemas IYR, PDR e IPYDCV.

No obstante, supóngase que se formula muy frecuentemente la siguiente consulta: buscar el número de copias C del repuesto R pedido en el contrato I. Esta consulta necesita una reunión de las relaciones descompuestas IYR e IPYDCV (o PDR e IPYDCV), mientras que se puede responder directamente mediante la relación Contratos. El coste añadido de esta consulta podría persuadirnos de conformarnos con un diseño en 3FN y no seguir descomponiendo Contratos.

13.8.2 Desnormalización

Puede que los motivos que impulsan a conformarse con una forma normal más débil lleven a dar un paso todavía más extremo: introducir de manera deliberada algo de redundancia. Por ejemplo, considérese la relación Contratos, que se halla en 3FN. Ahora bien, supóngase que una consulta frecuente es comprobar si el valor de un contrato es menor que el presupuesto del departamento que lo firma. Puede que se decida añadir a Contratos un campo para presupuesto S. Dado que idd es una clave de Departamentos, ahora se tiene en Contratos la dependencia $D \rightarrow S$, lo que significa que Contratos ya no se halla en 3FN. Pese a todo, puede que se decida seguir adelante con este diseño si la consulta que lo ha motivado es lo bastante

²Si este esquema parece complicado, téngase en cuenta que las situaciones de la vida real suelen requerir esquemas considerablemente más complejos.

importante. Este tipo de decisiones es evidentemente subjetivo, y se adopta al precio de una significativa redundancia.

13.8.3 Elección de la descomposición

Considérese nuevamente la relación Contratos. Hay varias opciones para tratar con la redundancia de esta relación:

- Se puede dejar Contratos tal y como está y aceptar la redundancia asociada con que esté en 3FN en vez de en FNBC.
- Se puede decidir que se desea evitar las anomalías resultantes de esta redundancia descomponiendo Contratos en FNBC mediante uno de los métodos siguientes:
 - Se tiene una descomposición por reunión sin pérdida en InfoRepuestos, con los atributos PDR, e InfoContratos, con los atributos IPYDCV. Como ya se ha observado, esta descomposición no conserva las dependencias, y hacer que las conserve exigiría que se añadiera una tercera relación, IYR, cuya única finalidad es permitir que se compruebe la dependencia $YR \rightarrow I$.
 - Se puede decidir sustituir Contratos sólo por InfoRepuestos e InfoContratos, aunque esta descomposición no conserve las dependencias.

Sustituir Contratos sólo por InfoRepuestos e InfoContratos no evita que se deba hacer cumplir la restricción $YR \rightarrow I$; sólo lo hace más costoso. Se podría crear un aserto en SQL-92 para que comprobara esta restricción:

```
CREATE ASSERTION compruebaDep
  CHECK ( NOT EXISTS
    ( SELECT *
        FROM InfoRepuestos IR, InfoContratos IC
        WHERE IR.idproveedor=IC.idproveedor
          AND IR.iddep=IC.iddep
      GROUP BY IC.id proyecto, IR.id repuesto
      HAVING COUNT (idc) > 1 ) )
```

Este aserto es costoso de evaluar, ya que implica una reunión seguida de una ordenación (para realizar la agrupación). En comparación, el sistema puede comprobar si YR es una clave principal de la tabla IYR manteniendo un índice para YR . Esta diferencia de coste en la comprobación de la integridad es la razón de la conservación de las dependencias. Por otro lado, si las actualizaciones son infrecuentes, puede que este mayor coste sea aceptable; por tanto, puede que se decida no mantener la tabla IYR (y, con bastante probabilidad, un índice para ella).

Como ejemplo adicional para ilustrar las opciones de descomposición, considérese nuevamente la relación Contratos y supóngase que también se tiene la restricción de integridad de que cada departamento utiliza cada proveedor, como máximo, para uno de sus proyectos: $PRC \rightarrow V$. Procediendo igual que antes, se tiene una descomposición por reunión sin pérdida de Contratos en PDR e IPYDCV. De manera alternativa, se podría empezar por emplear

la dependencia $PRC \rightarrow V$ para orientar la descomposición y sustituir Contratos por PRCV e IPYDRC. Luego se puede descomponer IPYDRC, orientados por $PD \rightarrow R$, para obtener PDR e IPYDC.

Ahora se tienen dos descomposiciones alternativas de Contratos por reunión sin pérdida en FNBC, ninguna de las cuales conserva las dependencias. La primera alternativa es sustituir Contratos por las relaciones PDR e IPYDCV. La segunda alternativa es sustituirla por PRCV, PDR e IPYDC. El añadido de IYR hace que la segunda descomposición (pero no la primera) conserve las dependencias. Una vez más, puede que el coste de mantener las tres relaciones IYR, PRCV e IPYDC (en vez de sólo IPYDCV) lleve a escoger la primera alternativa. En ese caso, hacer que se cumplan las DF dadas se vuelve más costoso. Se podría considerar la posibilidad de no hacer que se cumplan, pero esto comprometería la integridad de los datos.

13.8.4 Particiones verticales de las relaciones en FNBC

Supóngase que se ha decidido descomponer contratos en PDR e IPYDCV. Estos esquemas se hallan en FNBC y no hay ningún motivo para seguir descomponiéndolos más desde el punto de vista de la normalización. Sin embargo, supóngase que son muy frecuentes las consultas siguientes:

- Hallar los contratos que tiene el proveedor P.
- Hallar los contratos ofrecidos por el departamento D.

Estas consultas podrían llevarnos a descomponer IPYDCV en IP, ID e IYCV. Por supuesto, la descomposición no tiene pérdida y esas dos consultas tan importantes se pueden responder examinando relaciones de mucho menor tamaño. Otro motivo para considerar una posible descomposición es el control de los *puntos calientes* de la concurrencia. Si estas consultas son frecuentes, y las actualizaciones más frecuentes suponen el cambio de la cantidad (y del valor) de los productos implicados en los contratos, la descomposición mejora el rendimiento al reducir los conflictos de los bloqueos. Así, los bloqueos exclusivos se imponen principalmente sobre la tabla IYCV y las lecturas de IP e ID no entran en conflicto con esos bloqueos.

Siempre que se descompone una relación hay que tener en consideración las consultas que pueden verse perjudicadas por esa descomposición, especialmente si la única razón para llevarla a cabo es mejorar el rendimiento. Por ejemplo, si otra consulta importante tiene que hallar el valor total de los contratos de cada proveedor, implicará una reunión de las relaciones descompuestas IP e IYCV. En esta situación puede que se decida no llevar a cabo la descomposición.

13.8.5 Descomposición horizontal

Hasta ahora se ha tenido en cuenta sobre todo el modo de sustituir una relación por un conjunto de descomposiciones verticales. A veces merece la pena considerar si conviene sustituir una relación por dos relaciones que tienen los mismos atributos que la original, cada una de las cuales contiene un subconjunto de sus tuplas. De manera intuitiva, esta técnica resulta útil cuando se consultan de manera diferente diferentes subconjuntos de tuplas.

Por ejemplo, puede haber reglas diferentes para los grandes contratos, que se definen como contratos de valor superior a 10.000 €. (Quizás esos contratos se deban conceder mediante un proceso de subasta.) Esta restricción puede provocar cierto número de consultas en las que las tuplas de Contratos se seleccionen mediante una condición de la forma *valor* > 10.000. Un modo de abordar esta situación es crear un índice de árbol B+ agrupado por el campo *valor* de Contratos. De manera alternativa, se puede sustituir Contratos por dos relaciones denominadas GrandesContratos y PequeñosContratos, de significado evidente. Si esta consulta es el único motivo del índice, la descomposición horizontal ofrece todas las ventajas del índice sin la sobrecarga que supone su mantenimiento. Esta alternativa resulta especialmente atractiva si otras consultas importantes a Contratos necesitan también índices agrupados (para campos distintos de *valor*).

Si se sustituye Contratos por las dos relaciones GrandesContratos y PequeñosContratos, se puede ocultar este cambio definiendo una vista denominada Contratos:

```
CREATE VIEW Contratos(idc, idproveedor, id proyecto, iddep, idrepuesto, cantidad, valor)
AS ((SELECT *
      FROM    GrandesContratos)
UNION
  (SELECT *
      FROM    PequeñosContratos))
```

No obstante, cualquier consulta que sólo trabaje con GrandesContratos se debe expresar directamente sobre GrandesContratos y no sobre la vista. La expresión de la consulta sobre la vista Contratos con la condición de selección *valor* > 10.000 es equivalente a la expresión de la consulta sobre GrandesContratos, pero menos eficiente. Esto es bastante general: aunque se puedan ocultar las modificaciones del esquema conceptual añadiendo definiciones de vistas, los usuarios preocupados por el rendimiento tienen que ser conscientes de esas modificaciones.

Como ejemplo adicional, si Contratos tuviera otro campo más, *año*, y las consultas tuvieran que ver sobre todo con los contratos de un año determinado, se podría decidir dividir Contratos por años. Por supuesto, las consultas que tuvieran que ver con contratos de más de un año podrían exigir que se plantearan consultas a cada una de las relaciones descompuestas.

13.9 OPCIONES DE AJUSTE DE CONSULTAS Y DE VISTAS

El primer paso para ajustar una consulta es comprender el plan empleado por el SGBD para evaluarla. Los sistemas suelen ofrecer algún medio para identificar el plan empleado para evaluar las consultas. Una vez se comprende el plan seleccionado por el sistema, se puede considerar el modo de mejorar su rendimiento. Se puede considerar una selección de índices diferente o, quizás, la coagrupación de dos relaciones para las consultas de reunión, orientados por la comprensión del plan antiguo y de un plan mejor que se deseé que emplee el SGBD. Los detalles son parecidos a los del proceso inicial de diseño.

Conviene destacar que antes de crear índices nuevos se debe considerar si la reescritura de la consulta consigue resultados aceptables con los índices existentes. Por ejemplo, considérese la consulta siguiente con la conectiva OR:

```
SELECT E.númd
FROM Empleados E
WHERE E.afición='Sellos' OR E.edad=10
```

Si se dispone de índices tanto para *afición* como para *edad*, se pueden emplear para recuperar las tuplas necesarias, pero puede que el optimizador no logre reconocer esta oportunidad. Puede que el optimizador vea las condiciones de la cláusula **WHERE** en conjunto como no adecuadas para ninguno de los dos índices, haga una exploración secuencial de Empleados y aplique las selecciones sobre la marcha. Supóngase que se reescribe la consulta como unión de dos consultas, una con la cláusula **WHERE E.afición='Sellos'** y la otra con la cláusula **WHERE E.edad=10**. Ahora las dos consultas se responde de manera eficiente con la ayuda de los índices para *afición* y para *edad*.

También se debe considerar la reescritura de la consulta si se desean evitar determinadas operaciones costosas. Por ejemplo, la inclusión de **DISTINCT** en la cláusula **SELECT** hace que se eliminen los valores duplicados, lo que puede resultar costoso. Por tanto, se debe omitir **DISTINCT** siempre que sea posible. Por ejemplo, para una consulta a una sola relación se puede omitir **DISTINCT** siempre que se cumpla cualquiera de las condiciones siguientes:

- No importa la presencia de valores duplicados.
- Los atributos mencionados en la cláusula **SELECT** incluyen una posible clave de la relación.

A veces las consultas con **GROUP BY** y **HAVING** se pueden sustituir por consultas sin esas cláusulas, lo que elimina una operación de ordenación. Por ejemplo, considérese:

```
SELECT MIN (E.edad)
FROM Empleados E
GROUP BY E.númd
HAVING E.númd=102
```

Esta consulta es equivalente a

```
SELECT MIN (E.edad)
FROM Empleados E
WHERE E.númd=102
```

Las consultas complejas se escriben a menudo paso a paso, empleando relaciones temporales. Normalmente esas consultas se pueden reescribir sin la relación temporal para hacer que se ejecuten más rápido. Considérese la consulta siguiente para el cálculo del sueldo promedio de los departamentos dirigidos por Rupérez:

```
SELECT *
INTO Temp
FROM Empleados E, Departamentos D
WHERE E.númd=D.númd AND D.nombredir='Rupérez'

SELECT T.númd, AVG (T.sueldo)
FROM Temp T
GROUP BY T.númd
```

Esta consulta se puede reescribir como:

```
SELECT E.númd, AVG (E.sueldo)
FROM Empleados E, Departamentos D
WHERE E.númd=D.númd AND D.nombredir='Rupérez'
GROUP BY E.númd
```

La consulta reescrita no materializa la relación intermedia Temp y, por tanto, es probable que sea más rápida. De hecho, puede que el optimizador encuentre incluso un plan muy eficiente basado únicamente en los índices que no recupere nunca tuplas de empleados si hay un índice compuesto de árbol B+ para $\langle \text{númd}, \text{sueldo} \rangle$. Este ejemplo ilustra una observación de carácter general: *al reescribir las consultas para evitar elementos temporales innecesarios no sólo se evita crear las relaciones temporales, también se abren más posibilidades de optimización para que las explore el optimizador.*

No obstante, en algunas situaciones, si el optimizador no es capaz de hallar un buen plan para una consulta compleja (generalmente una consulta anidada con correlación), puede que merezca la pena reescribir esa consulta empleando relaciones temporales para orientar al optimizador hacia un buen plan.

De hecho, las consultas anidadas son un motivo frecuente de ineficiencia, ya que muchos optimizadores tienen problemas para tratarlas. Siempre que sea posible es mejor reescribir las consultas anidadas sin el anidamiento y las consultas correlacionadas sin la correlación. Como ya se ha indicado, puede que una buena reformulación de las consultas exija introducir nuevas relaciones temporales, y las técnicas para hacerlo de manera sistemática (en teoría debería hacerlo el optimizador) han sido objeto de amplios estudios. A menudo, sin embargo, es posible reescribir las consultas anidadas sin el anidamiento o emplear relaciones temporales.

13.10 CONSECUENCIAS DE LA CONCURRENCIA

En sistemas con muchos usuarios concurrentes se deben tener en consideración varias cosas más. Las transacciones obtienen *bloqueos* sobre las páginas a las que tienen acceso, y puede que otras transacciones queden bloqueadas mientras esperan los bloqueos sobre los objetos a los que desean tener acceso.

En el Apartado 8.5 se observó que se deben minimizar los retrazos en los bloqueos para conseguir un buen rendimiento y se identificaron dos maneras concretas de reducir los bloqueos:

- Reducir el tiempo durante el que las transacciones mantienen los bloqueos.
- Reducir los puntos calientes.

A continuación se analizarán las técnicas para conseguir esos objetivos.

13.10.1 Reducción de la duración de los bloqueos

Retraso de las peticiones de bloqueo. Se ajustan las transacciones escribiendo en variables locales del programa y aplazando las modificaciones de la base de datos hasta el final

de la transacción. Esto retrasa la adquisición de los bloqueos correspondientes y reduce el tiempo durante el que éstos se mantienen.

Aceleración de las transacciones. Cuanto antes se complete una transacción, antes liberará sus bloqueos. Ya se han examinado varias maneras de acelerar las consultas y las actualizaciones (por ejemplo, ajustando los índices o reescribiendo las consultas). Además, un reparto meticuloso de las tuplas de las relaciones y de sus índices asociados entre diferentes discos puede mejorar de manera significativa el acceso concurrente. Por ejemplo, si se tiene una relación en un disco y su índice en otro, los accesos al índice pueden llevarse a cabo sin interferir con los accesos a la relación, al menos, en el nivel de las operaciones de lectura de disco.

Sustitución de las transacciones largas por otras más cortas. A veces, se lleva a cabo demasiado trabajo dentro de la transacción, tarda mucho tiempo y retiene demasiado los bloqueos. Se puede considerar la posibilidad de reescribir la transacción como dos o más transacciones de menor tamaño; los cursos retenibles (véase el Apartado 6.1.2) pueden resultar útiles para ello. La ventaja es que cada una de las nuevas transacciones se completa más rápidamente y libera antes sus bloqueos. El inconveniente es que la lista original de operaciones ya no se ejecuta atómicamente, y el código de la aplicación deberá afrontar situaciones en las que falle alguna de las transacciones nuevas.

Creación de almacenes. Las consultas complejas pueden retener mucho tiempo los bloqueos compartidos. A menudo, no obstante, esas consultas incorporan el análisis estadístico de tendencias comerciales y resulta aceptable ejecutarlas sobre una copia de los datos que esté ligeramente desactualizada. Esto ha provocado la popularidad de los *almacenes de datos*, que son bases de datos que complementan la base de datos operativa conservando una copia de los datos que se emplean en las consultas complejas (Capítulo 16). La ejecución de estas consultas sobre el almacén de datos libera a la base de datos operativa de la carga de las consultas de larga duración.

Consideración de niveles inferiores de aislamiento. En muchos casos, como las consultas que generan información agregada o resúmenes estadísticos, se pueden emplear niveles inferiores de aislamiento de SQL como REPEATABLE READ o READ COMMITTED (Apartado 8.6). Los niveles inferiores de aislamiento provocan menores sobrecargas de bloqueos, y el diseñador de la aplicación debe lograr buenos equilibrios de diseño.

13.10.2 Reducción de los puntos calientes

Retraso de las operaciones en puntos calientes. Ya se ha analizado la conveniencia del retraso de las solicitudes de bloqueo. Evidentemente, esto resulta especialmente importante para las peticiones que implican a objetos que se utilizan con frecuencia.

Optimización de las pautas de acceso. La *pauta* de las actualizaciones de una relación también puede resultar significativa. Por ejemplo, si se insertan tuplas en la relación Empleados según el orden de *ide* y se tiene un índice de árbol B+ para *ide*, cada inserción irá a la última página de la hoja del árbol B+. Esto provoca puntos calientes a lo largo de la ruta desde la raíz a la página de la hoja situada más a la derecha. Estas consideraciones pueden hacer que se prefiera un índice asociativo a un índice de árbol B+ o a la creación de un índice sobre un campo diferente. Obsérvese que esta pauta de acceso también provoca

un bajo rendimiento para los índices ISAM, ya que la última página de la hoja pasa a ser un punto caliente. No se trata de un problema para los índices asociativos, ya que el proceso asociativo elige de manera aleatoria el lugar en que se inserta cada registro.

Operaciones de reparto para los puntos calientes. Considérese una transacción de entrada de datos que añade nuevos registros a un archivo (por ejemplo, inserciones en una tabla guardada como archivo de pila). En lugar de añadir un registro por cada transacción y obtener un bloqueo de la última página para cada registro, se puede sustituir esta transacción por varias transacciones, cada una de las cuales escriba los registros en un archivo local y añada de manera periódica un lote de registros al archivo principal. Aunque el trabajo total sea mayor, esto reduce el conflicto entre bloqueos para la última página del archivo original.

Como ejemplo adicional de reparto, supóngase que se realiza un seguimiento del número de registros insertados en un contador. En lugar de actualizar este contador una vez por cada registro, el enfoque anterior da lugar a que se actualicen varios contadores y se actualice periódicamente el contador principal. Esta idea se puede adaptar a muchos usos de los contadores, con diversos grados de esfuerzo. Por ejemplo, considérese un contador que realiza el seguimiento del número de reservas, con la regla de que sólo se permite una nueva reserva si el contador se halla por debajo de un valor máximo. Este contador se puede sustituir por tres contadores, cada uno con un tercio del umbral máximo original, y tres transacciones que empleen esos contadores en lugar del original. Se obtiene una mayor concurrencia, pero hay que afrontar el caso de que uno de los contadores haya alcanzado su valor máximo pero algún otro todavía se pueda incrementar. Por tanto, el precio de la mayor concurrencia es una mayor complejidad de la lógica del código de la aplicación.

Selección de índices. Si una relación se actualiza frecuentemente, los índices de árboles B+ pueden convertirse en un cuello de botella para el control de la concurrencia, ya que todos los accesos mediante el índice deben pasar por la raíz. Por tanto, la raíz y las páginas del índice inmediatamente por debajo de ella pueden transformarse en puntos calientes. Si el SGBD emplea protocolos de bloqueo especializados para los índices de árbol y, en especial, define bloqueos de granularidad fina, este problema se palía en gran medida. Muchos sistemas actuales emplean técnicas de este tipo.

Pese a todo, esta consideración puede hacer que se elija un índice ISAM en algunas situaciones. Como los niveles de índices de los índices ISAM son estáticos, no hace falta obtener bloqueos para esas páginas; sólo hace falta bloquear las páginas de las hojas. Los índices ISAM pueden resultar preferibles a los índices de árboles B+, por ejemplo, si tienen lugar actualizaciones frecuentes pero se espera que la distribución relativa de los registros y su número (y tamaño) para un rango dado de valores de la clave de búsqueda permanezcan aproximadamente iguales. En ese caso, el índice ISAM ofrece una menor sobrecarga de bloqueos (y un menor conflicto entre bloqueos), y la distribución de los registros es tal que se crean pocas páginas de desbordamiento.

Los índices asociativos no crean esos cuellos de botella para la concurrencia, a menos que la distribución de los datos esté muy sesgada y muchos elementos de datos estén concentrados en unos pocos lugares. En ese caso, las entradas de directorio para esos lugares pueden transformarse en puntos calientes.

13.11 ESTUDIO DE UN CASO: LA TIENDA EN INTERNET

Volviendo al caso de estudio en desarrollo, TiposBD considera la carga de trabajo esperada para la librería B&N. El propietario de la librería espera que la mayor parte de sus clientes busquen los libros por su código ISBN antes de formular el pedido. Formular un pedido implica insertar un registro en la tabla Pedidos y uno o varios registros en la relación ListaPedidos. Si se dispone de un número suficiente de libros, se prepara el envío y se define un valor para *fecha_envío* en la relación ListaPedidos. Además, la cantidad de libros disponible en existencias cambia continuamente, ya que se formulan pedidos que disminuyen la cantidad disponible y llegan nuevos libros de los proveedores que la aumentan.

El equipo de TiposBD comienza considerando la búsqueda de los libros por su ISBN. Dado que *isbn* es una clave, cada consulta de igualdad sobre *isbn* devuelve como máximo un registro. Por tanto, para acelerar las consultas de los clientes que buscan los libros con un ISBN dado, TiposBD decide crear un índice asociativo no agrupado para *isbn*.

A continuación consideran las actualizaciones de las cantidades de los diferentes libros. Para actualizar el valor de *stock* para un libro, primero hay que buscar el libro por su ISBN; el índice para *isbn* acelera este proceso. Dado que el valor de *stock* de cada libro se actualiza con bastante frecuencia, TiposBD considera también dividir verticalmente la relación Libros en las dos relaciones siguientes:

CantLibros(isbn, cantidad)
RestoLibros(isbn, título, autor, precio, año_publicación)

Por desgracia, esta división vertical hace más lenta otra consulta muy popular: la consulta de igualdad del ISBN para recuperar toda la información sobre cada libro exige ahora la reunión de *CantLibros* y *RestoLibros*. Por tanto, TiposBD decide no dividir Libros verticalmente.

TiposBD cree que es posible que los clientes también deseen buscar los libros por su título y su autor, y decide añadir índices asociativos no agrupados para *título* y para *autor* —estos índices no resultan costosos de mantener, ya que el conjunto de libros cambia rara vez, pese a que las existencias de cada libro varíen a menudo—.

A continuación, TiposBD considera la relación Clientes. Cada cliente se identifica en primer lugar por su número de identificación de cliente, que es único. Por tanto, las consultas más frecuentes a Clientes son las de igualdad relativas al número de identificación de cliente, y TiposBD decide crear un índice asociativo agrupado para *idc* para conseguir la velocidad máxima para esa consulta.

Pasando a la relación Pedidos, TiposBD ve que está implicada en dos consultas: la inserción de pedidos nuevos y la recuperación de los ya existentes. Las dos consultas implican al atributo *númpedido* como clave de búsqueda y, por tanto, TiposBD decide crearle un índice. ¿Qué tipo de índice resulta más conveniente —de árbol B+ o uno asociativo—? Dado que los números de pedido se asignan secuencialmente y corresponden a la fecha del pedido, la ordenación por *númpedido* ordena también de manera efectiva por fecha de pedido. Por tanto, TiposBD decide crear un índice de árbol B+ agrupado para *númpedido*. Aunque los requisitos operativos mencionados hasta ahora no favorecen ni a los índices B+ ni a los asociativos, probablemente B&N desee controlar las actividades diarias, y el árbol B+ agrupado resulta una mejor opción

para ese tipo de consultas por rangos. Por supuesto, eso significa que la recuperación de todos los pedidos de un cliente dado puede resultar costosa para los clientes con muchos pedidos, ya que la agrupación por *númpedido* impide la agrupación por otros atributos, como el *idc*.

La relación ListaPedidos implica, sobre todo, inserciones, con la actualización ocasional de una fecha de envío o la consulta para averiguar todos los componentes de un pedido dado. Si ListaPedidos sigue ordenada por *númpedido*, todas las inserciones serán añadidas al final de la relación y, por tanto, muy eficientes. El índice B+ agrupado para *númpedido* mantiene este orden y también acelera la recuperación de todos los elementos de cada pedido. Para actualizar la fecha de un envío hay que buscar las tuplas por *númpedido* y por *isbn*. El índice para *númpedido* también resulta de ayuda en este caso. Aunque sería mejor a este efecto un índice para $\langle \text{númpedido}, \text{isbn} \rangle$, las inserciones no resultarían tan eficientes como con un índice sólo para *númpedido*; TiposBD, por tanto, decide indexar ListaPedidos sólo para *númpedido*.

13.11.1 Ajuste de la base de datos

Varios meses después del lanzamiento del sitio web de B&N se convoca a TiposBD y se le comunica que las preguntas de los clientes sobre los pedidos pendientes se están procesando muy lentamente. B&N ha obtenido mucho éxito y las tablas Pedidos y ListaPedidos se han hecho enormes.

Pensando más sobre el diseño, TiposDB se da cuenta de que hay dos tipos de pedidos: *pedidos completados*, para los que ya se han enviado todos los libros, y *pedidos completados parcialmente*, para los que todavía hay que enviar algún libro. La mayor parte de las peticiones de búsqueda de pedidos por parte de los consumidores tiene que ver con los pedidos completados parcialmente, que son una pequeña parte de todos los pedidos. Por tanto, TiposBD decide dividir horizontalmente las tablas Pedidos y ListaPedidos de acuerdo con *númpedido*. Esto da lugar a cuatro relaciones nuevas: PedidosNuevos, PedidosAntiguos, ListaPedidosNuevos y ListaPedidosAntiguos.

Cada pedido y sus componentes se hallan exactamente en un par de relaciones —y se puede determinar ese par, antiguo o nuevo, mediante una mera comprobación de *númpedido*— y las consultas que tienen que ver con ese pedido siempre se pueden evaluar empleando sólo las relaciones correspondientes. Ahora resultan más lentas algunas consultas, como las que solicitan todos los pedidos de un usuario, ya que exigen que se examinen dos conjuntos de relaciones. Sin embargo, esas consultas son infrecuentes y su rendimiento es aceptable.

13.12 PRUEBAS DE RENDIMIENTO DE LOS SGBD

Hasta ahora se ha considerado la manera de mejorar el diseño de las bases de datos para obtener un mejor rendimiento. A medida que las bases de datos crecen, sin embargo, puede que el SGBD subyacente deje de poder ofrecer un rendimiento adecuado, incluso con el mejor diseño posible, y haya que considerar la posibilidad de mejorar el sistema, generalmente mediante la adquisición de hardware más rápido y de memoria adicional. Puede que también se considere la posibilidad de migrar la base de datos a un nuevo SGBD.

Al evaluar los productos de SGBD el rendimiento es una consideración importante. Los SGBD son piezas de software complejas, y los distintos fabricantes pueden dirigir sus sistemas

a diferentes segmentos del mercado dedicándose más a optimizar determinadas partes del sistema o escogiendo distintos diseños del sistema. Por ejemplo, algunos sistemas se diseñan para que ejecuten eficientemente consultas complejas, mientras que otros están diseñados para ejecutar muchas transacciones sencillas por segundo. Dentro de cada categoría de estos sistemas hay muchos productos competidores. Para ayudar a los usuarios a escoger el SGBD más adecuado a sus necesidades se han desarrollado varias **pruebas de rendimiento**. Estas pruebas incluyen medidas del rendimiento de una determinada clase de aplicaciones (por ejemplo, las pruebas TPC) y medidas para saber cómo realiza el SGBD diversas operaciones (por ejemplo, la prueba Wisconsin).

Las pruebas deben ser trasladables, fáciles de comprender y aplicables con naturalidad a ejemplares experimentales de mayor tamaño. Deben medir el *rendimiento máximo* (por ejemplo, *transacciones por segundo*, o *tps*), y las *relaciones precio/rendimiento* (es decir, $\text{€}/\text{tps}$) de las cargas de trabajo típicas de un dominio de aplicación dado. El Consejo de procesamiento de transacciones (Transaction Processing Council, TPC) se creó para definir pruebas de rendimiento para el procesamiento de transacciones y los sistemas de bases de datos. Investigadores académicos y organizaciones industriales han propuesto otras pruebas de rendimiento bien conocidas. Las pruebas de rendimiento que son exclusivas de un determinado fabricante no resultan muy útiles para comparar diferentes sistemas (aunque puedan resultar útiles para determinar el modo en que un sistema dado tratará una carga de trabajo determinada).

13.12.1 Pruebas de rendimiento de SGBD bien conocidas

Pruebas de rendimiento del procesamiento de transacciones en línea. Las pruebas de rendimiento TPC-A y TPC-B constituyen las definiciones normalizadas de las medidas *tps* y $\text{€}/\text{tps}$. TPC-A mide el rendimiento y el precio de una red informática, además del SGBD, mientras que la prueba de rendimiento TPC-B considera el SGBD por sí mismo. Estas pruebas de rendimiento implican una transacción sencilla que actualiza tres registros de datos, de tres tablas diferentes, y añade un registro a una cuarta tabla. Se especifican rigurosamente varios detalles (por ejemplo, la distribución de llegadas de las transacciones, el método de interconexión o las propiedades del sistema), lo que garantiza que los resultados para diferentes sistemas se puedan comparar con sentido. La prueba de rendimiento TPC-C es un conjunto más complejo de tareas transaccionales que TPC-A y TPC-B. Modela un almacén que realiza un seguimiento de los elementos suministrados a los clientes e implica cinco tipos de transacciones. Cada transacción de TPC-C resulta mucho más costosa que las transacciones de TPC-A o de TPC-B, y TPC-C ejercita un rango mucho más amplio de capacidades del sistema, como el empleo de índices secundarios y el aborto de transacciones. Prácticamente ha sustituido a TPC-A y a TPC-B como prueba de rendimiento normalizada para el procesamiento de transacciones.

Pruebas de rendimiento para consultas. La prueba de rendimiento Wisconsin se emplea mucho para medir el rendimiento de consultas relacionales sencillas. La prueba de rendimiento Set Query mide el rendimiento de un conjunto de consultas más complejas y la prueba de rendimiento *AS³AP* mide el rendimiento de una carga de trabajo mixta de transacciones, consultas relacionales y funciones de utilidad. La prueba de rendimiento TPC-D es un conjunto de consultas de SQL complejas que se pretende que sea representativo del dominio

de aplicaciones de ayuda a la toma de decisiones. El Consejo OLAP desarrolló también una prueba de rendimiento para las consultas complejas de ayuda a la toma de decisiones, que incluye algunas consultas que no se pueden expresar con facilidad en SQL; está pensada para medir sistemas para el *procesamiento analítico en línea (online analytic processing, OLAP)*, que se analiza en el Capítulo 16, más que para los sistemas de SQL tradicionales. La prueba de rendimiento Sequoia 2000 está diseñada para comparar el soporte de los SGBD a los sistemas de información geográfica.

Pruebas de rendimiento para bases de datos orientadas a objetos. Las pruebas de rendimiento 001 y 007 miden el rendimiento de los sistemas de bases de datos orientados a objetos. La prueba de rendimiento Bucky mide el rendimiento de los sistemas de bases de datos relacionales orientados a objetos. (Los sistemas de bases de datos orientados a objetos se analizan en el Capítulo 15.)

13.12.2 Empleo de pruebas de rendimiento

Las pruebas de rendimiento deben emplearse con una profunda comprensión de para qué medida se han diseñado y del entorno de aplicación en el que se debe utilizar el SGBD. Cuando se emplean pruebas de rendimiento para orientar la elección de SGBD hay que tener presentes las siguientes directrices:

- **¿Es muy significativa una determinada prueba de rendimiento?** Las pruebas de rendimiento que intentan traducir el rendimiento en un solo número pueden resultar excesivamente simplistas. Los SGBD son elementos complejos de software que se emplean en gran variedad de aplicaciones. Una buena prueba de rendimiento debe tener un conjunto de tareas que estén cuidadosamente escogidas para abarcar un dominio de aplicación concreto y probar las características del SGBD importantes para ese dominio.
- **¿Refleja bien la carga de trabajo la prueba de rendimiento?** Hay que considerar la carga de trabajo esperada y compararla con la prueba de rendimiento. Se debe dar más peso al rendimiento de esas tareas de la prueba de rendimiento (por ejemplo, consultas y actualizaciones) que son parecidas a las tareas importantes de la carga de trabajo. También hay que considerar el modo en que se calculan las cifras de las pruebas de rendimiento. Por ejemplo, el tiempo de ejecución de cada consulta puede conducir a error si se considera en una configuración multiusuario: puede que un sistema tenga tiempos de ejecución mayores debido a una E/S más lenta. Para una carga de trabajo multiusuario, y con suficientes discos para una E/S paralela, un sistema podría superar en rendimiento a otro con un tiempo menor de ejecución.
- **Crear una prueba de rendimiento propia.** Los fabricantes ajustan a menudo los sistemas *ad hoc* para obtener buenas cifras en las pruebas de rendimiento importantes. Para contrarrestar este efecto se puede crear una prueba de rendimiento propia modificando ligeramente las pruebas de rendimiento normalizadas o sustituyendo las tareas de una prueba de rendimiento normalizada por tareas que sean parecidas a las necesidades particulares.

13.13 PREGUNTAS DE REPASO

Las respuestas a las preguntas de repaso se pueden encontrar en los apartados indicados.

- ¿Cuáles son los componentes de la descripción de una carga de trabajo? (**Apartado 13.1.1**)
- ¿Qué decisiones hay que tomar durante el proceso de diseño físico? (**Apartado 13.1.2**)
- Describanse seis directrices de alto nivel para la selección de índices. (**Apartado 13.2**)
- ¿Cuándo se deben crear índices agrupados? (**Apartado 13.4**)
- ¿Qué es la coagrupación y cuándo se debe utilizar? (**Apartado 13.4.1**)
- ¿Qué es un plan exclusivo para índices y cómo se crean los índices para los planes exclusivos para índices? (**Apartado 13.5**)
- ¿Por qué es un problema difícil el ajuste automático de los índices? Dese un ejemplo. (**Apartado 13.6.1**)
- Dese un ejemplo de algoritmo para el ajuste automático de índices. (**Apartado 13.6.2**)
- ¿Por qué es importante el ajuste de las bases de datos? (**Apartado 13.7**)
- ¿Cómo se ajustan los índices, el esquema conceptual, y las consultas y las vistas? (**Apartados 13.7.1 a 13.7.3**)
- ¿Cuáles son las opciones para el ajuste del esquema conceptual? ¿Qué son las técnicas siguientes y cuándo se deben aplicar?: aceptación de una forma normal más débil, desnormalización y descomposiciones horizontal y vertical. (**Apartado 13.8**)
- ¿Qué opciones hay para el ajuste de consultas y vistas? (**Apartado 13.9**)
- ¿Cuáles son los efectos de los bloqueos en el rendimiento de las bases de datos? ¿Cómo se pueden reducir los conflictos entre bloqueos y los puntos calientes? (**Apartado 13.10**)
- ¿Por qué hay pruebas de rendimiento normalizadas para las bases de datos y qué métricas habituales se emplean para evaluar los sistemas de bases de datos? Describanse unas cuantas pruebas de rendimiento de bases de datos populares (**Apartado 13.12**)

EJERCICIOS

Ejercicio 13.1 Considérese el siguiente esquema en FNBC para una parte de base de datos corporativa sencilla (la información sobre los tipos no es de relevancia para esta pregunta y se omite):

Emp (*ide, nombree, dir, sueldo, edad, años, iddep*)
 Dep (*idd, nombred, piso, presupuesto*)

Supóngase que se conoce que las siguientes consultas son las seis más frecuentes de la carga de trabajo para esta empresa y que las seis son aproximadamente equivalentes en frecuencia e importancia:

- Mostrar el identificador, el nombre y la dirección de los empleados de un rango de edad especificado por el usuario.

- Mostrar el identificador, el nombre y la dirección de los empleados que trabajan en el departamento de nombre especificado por el usuario.
 - Mostrar el identificador y la dirección de los empleados de nombre especificado por el usuario.
 - Mostrar el sueldo medio global de los empleados.
 - Mostrar el sueldo medio de los empleados por edades; es decir, por cada edad presente en la base de datos, indicar esa edad y el sueldo medio correspondiente.
 - Mostrar toda la información de los departamentos, ordenada por número de piso del departamento.
1. Dada esta información, y suponiendo que esas consultas son más importantes que cualquier actualización, diséñese un esquema físico para la base de datos corporativa que dé un buen rendimiento para la carga de trabajo esperada. En concreto, decídase qué atributos se indexarán y si cada uno de esos índices debe ser agrupado o no. Supóngase que los índices de árboles B+ son el único tipo de índice soportado por el SGBD y que se permiten claves tanto de un solo atributo como de varios. Específiquese el diseño físico identificando los atributos que se recomienda indexar mediante árboles B+ agrupados o no agrupados.
 2. Rediséñese el esquema físico suponiendo que el conjunto de consultas importantes se cambia por el siguiente:
 - Muéstrense el identificador y la dirección de los empleados de nombre especificado por el usuario.
 - Muéstrese el sueldo máximo global de los empleados.
 - Muéstrese el sueldo medio de los empleados por departamentos; es decir, para cada valor de *iddep*, indíquese el valor de *iddep* y el sueldo medio de los empleados de ese departamento.
 - Muéstrese la suma de los presupuestos de todos los departamentos por pisos; es decir, indíquese el piso y la suma.
 - Supóngase que hay que ajustar esta carga de trabajo con un asistente para el ajuste automático de índices. Esbózense las principales etapas de la ejecución del algoritmo de ajuste del índice y el conjunto de configuraciones posibles que habría que considerar.

Ejercicio 13.2 Considérese el siguiente esquema relacional en FNBC para una parte de una base de datos de una universidad (la información sobre los tipos no tiene relevancia para esta pregunta y se omite):

Prof(dni, nombrep, despacho, edad, sexo, especialidad, idd_dep)
Dep(idd, nombred, presupuesto, num_doctorandos, dni_director)

Supóngase que se sabe que las consultas siguientes son las cinco más frecuentes de la carga de trabajo para esta universidad y que las cinco son aproximadamente equivalentes en frecuencia y en importancia:

- Mostrar el nombre, edad y despacho de los profesores del sexo (masculino o femenino) especificado por el usuario que tienen la especialidad de investigación especificada por el usuario (por ejemplo, *procesamiento recursivo de consultas*). Supóngase que la universidad tiene un claustro de profesores diversificado, lo que hace muy improbable que muchos profesores tengan la misma especialidad de investigación.
- Mostrar toda la información departamental para los departamentos con profesores en el rango de edad especificado por el usuario.
- Mostrar el identificador, el nombre y el nombre del director de los departamentos con el número de doctorandos especificado por el usuario.
- Mostrar el presupuesto más bajo de entre los departamentos de la universidad.
- Mostrar toda la información sobre los profesores que son directores de departamento.

Estas consultas tienen lugar mucho más frecuentemente que las actualizaciones, por lo que se deben crear los índices necesarios para acelerarlas. No obstante, no se debe crear ningún índice innecesario, ya que pueden tener lugar actualizaciones (y se verían frenadas por los índices innecesarios). Dada esta información, diséñese un esquema físico para la base de datos de la universidad que dé un buen rendimiento para la carga de trabajo esperada. En especial, decídase qué atributos hay que indexar y si cada índice debe estar agrupado o no. Supóngase que el SGBD soporta tanto los índices de árboles B+ como los asociativos y que se permiten claves de búsqueda tanto de un solo atributo como de varios.

1. Especíquese el diseño físico identificando los atributos para los que se recomienda la indexación, indicando si cada índice debe estar agrupado o no y si debe ser un índice de árbol B+ o asociativo.
2. Supóngase que hay que ajustar esta carga de trabajo con un asistente para el ajuste automático de índices. Esbócense las principales etapas del algoritmo y el conjunto de posibles configuraciones considerado.
3. Rediseñese el esquema físico, suponiendo que el conjunto de consultas importantes se cambia por el siguiente:
 - Mostrar el número de especialidades diferentes abarcadas por los profesores de cada departamento, por departamentos.
 - Averiguar el departamento con menor número de doctorandos.
 - Averiguar el profesor más joven que es director de departamento.

Ejercicio 13.3 Considérese el siguiente esquema relacional en FNBC para una parte de la base de datos de una empresa (la información sobre los tipos no tiene relevancia para esta pregunta y se omite):

Proyecto(*nup, nombre_proy, dep_prom_proy, resp_proy, tema, presupuesto*)

Responsable(*idr, nombre_resp, dep_resp, sueldo, edad, sexo*)

Obsérvese que cada proyecto lo promueve algún departamento, cada responsable está empleado en algún departamento y el responsable de un proyecto no necesita estar empleado en el mismo departamento (que promueve el proyecto). Supóngase que se sabe que las consultas siguientes son las cinco más frecuentes de la carga de trabajo de esta empresa y que las cinco son aproximadamente equivalentes en frecuencia e importancia:

- Mostrar el nombre, edad y sueldo de los responsables del sexo (masculino o femenino) especificado por el usuario que trabajan en un departamento dado. Se puede suponer que, aunque hay muchos departamentos, cada uno de ellos alberga muy pocos responsables de proyecto.
- Mostrar el nombre de todos los proyectos con responsables cuya edad se halla en un rango especificado por el usuario (por ejemplo, menor de 30).
- Mostrar el nombre de todos los departamentos en los que un responsable de ese departamento dirige un proyecto promovido por ese departamento.
- Mostrar el nombre del proyecto con menor presupuesto.
- Mostrar el nombre de todos los responsables del mismo departamento que un proyecto dado.

Estas consultas tienen lugar mucho más frecuentemente que las actualizaciones, por lo que se deben crear los índices que hagan falta para acelerarlas. No obstante, no se debe crear ningún índice innecesario, ya que se producirán actualizaciones (y se verían frenadas por los índices innecesarios). Dada esta información, diseñese un esquema físico para la base de datos de la empresa que dé un buen rendimiento para la carga de trabajo esperada. En concreto, decidase qué atributos deben indexarse y si cada índice debe ser agrupado o no. Supóngase que el SGBD soporta tanto índices de árboles B+ como asociativos y que se permiten tanto claves de índice de un solo atributo como de varios.

1. Especíquese el diseño físico identificando los atributos para los que se recomienda la indexación, indicando si cada uno de los índices debe ser agrupado o no y si debe ser un índice de árbol B+ o asociativo.
2. Supóngase que hay que ajustar esta carga de trabajo con un asistente para el ajuste automático de índices. Esbócense las principales etapas del algoritmo y el conjunto de posibles configuraciones considerado.
3. Rediseñese el esquema físico suponiendo que el conjunto de consultas importantes se cambia por el siguiente:
 - Averiguar el total de los presupuestos de los proyectos gestionados por un mismo responsable; es decir, mostrar *resp_proy* y el total de los presupuestos gestionados por ese responsable, para todos los valores de *resp_proy*.
 - Averiguar el total de los presupuestos de los proyectos gestionados por cada responsable pero sólo para los responsables que se hallan en el rango de edad especificado por el usuario.

- Averiguar el número de responsables masculinos.
- Averiguar la edad media de los responsables.

Ejercicio 13.4 El Club de los Trotamundos está organizado en capítulos. El presidente de un capítulo no puede actuar nunca como presidente de ningún otro capítulo, y cada capítulo paga un sueldo a su presidente. Los capítulos no dejan de trasladarse a ubicaciones nuevas y se elige un presidente nuevo cuando (y sólo cuando) el capítulo se traslada. Estos datos se guardan en la relación $T(C,S,U,P)$, donde los atributos son los capítulos (C), los sueldos (S), las ubicaciones (U) y los presidentes (P). Se formulan frecuentemente consultas de la forma siguiente, y se deben poder contestar sin calcular ninguna reunión: “¿Quién era el presidente del capítulo X cuando se hallaba en la ubicación Y ? ”

1. Mostrar las DF enunciadas que T debe cumplir.
2. ¿Cuáles son las posibles claves de la relación T ?
3. ¿En qué forma normal se halla el esquema T ?
4. Diséñese un buen esquema de base de datos para el club. (Recuérdese que el diseño *debe* satisfacer el requisito de la consulta formulada.)
5. ¿En qué forma normal se halla este buen esquema? Dese un ejemplo de consulta que probablemente se ejecute más lentamente con este esquema que con la relación T .
6. ¿Hay alguna descomposición de T en FNBC por reunión sin pérdida que conserve las dependencias?
7. ¿Hay alguna vez buenas razones para aceptar algo menos que 3FN al diseñar un esquema para una base de datos relacional? Empléese este ejemplo, si es necesario añadiéndole alguna restricción, para ilustrar la respuesta.

Ejercicio 13.5 Considérese la siguiente relación en FNBC que muestra el identificador, el tipo (por ejemplo, tuerca o tornillo) y el coste de diferentes repuestos, junto con el número disponible o en existencias:

Repuestos (*idr*, *nombrer*, *coste*, *cantidad*)

Las dos consultas siguientes son extremadamente importantes:

- Averiguar el número total de unidades disponibles por tipo de repuesto, para todos los tipos. (Es decir, la suma de los valores de *num_disp* de todas las tuercas, la de todos los tornillos, etcétera.)
 - Mostrar la *idr* de los repuestos de mayor coste.
1. Describábase el diseño físico que se escogería para esta relación. Es decir, ¿qué tipo de estructura de archivos se escogería para el conjunto de registros de Repuestos y qué índices se crearían?
 2. Supóngase que los clientes se quejan posteriormente de que el rendimiento sigue sin ser satisfactorio (dados los índices y la organización de archivos escogidos para la relación Repuestos en respuesta a la pregunta anterior). Dado que no se puede permitir la adquisición de hardware ni software nuevos, hay que considerar el rediseño del esquema. Explíquese el modo de obtener mejor rendimiento describiendo el esquema de relaciones que se utilizaría y la elección de las organizaciones de archivos y de los índices para esas relaciones.
 3. ¿Cómo cambiarían las respuestas a esas dos preguntas, si es que lo harían, si el sistema no soportara índices con claves de búsqueda con varios atributos?

Ejercicio 13.6 Considérense las siguientes relaciones en FNBC, que describen a los empleados y a los departamentos en los que trabajan:

Emp (*ide*, *sueldo*, *idd*)

Dep (*idd*, *ubicación*, *presupuesto*)

Las consultas siguientes son extremadamente importantes:

- Averiguar la ubicación en que trabaja el empleado especificado por el usuario.

- Comprobar si el presupuesto de un departamento es mayor que el sueldo de cada empleado de ese departamento.
1. Describirse el diseño físico que se escogería para esta relación. Es decir, ¿qué tipo de estructura de archivos se elegiría para estas relaciones y qué índices se crearían?
 2. Supóngase que los clientes se quejan posteriormente de que el rendimiento sigue sin ser satisfactorio (dados los índices y la organización de los archivos que se escogieron para las relaciones en respuesta a la pregunta anterior). Dado que no se puede permitir la adquisición de nuevo hardware ni software, hay que considerar el rediseño del esquema. Explíquese la manera en que se intentaría obtener mejor rendimiento describiendo el esquema de relaciones que se emplearía, y la elección de organizaciones de archivos y de índices para esas relaciones.
 3. Supóngase que el sistema de bases de datos tiene implementaciones muy inefficientes de las estructuras de índices. ¿Qué clase de diseño se probaría en ese caso?

Ejercicio 13.7 Considérense las siguientes relaciones en FNBC, que describen los departamentos y los empleados de una empresa:

```
Dep(idd, nombred, ubicación, idresponsable)
Emp(ide, sueldo)
```

Las consultas siguientes son extremadamente importantes:

- Mostrar el nombre e identificador de los responsables de cada departamento de la ubicación especificada por el usuario, en orden alfabético de nombres de departamento.
 - Averiguar el sueldo medio de los empleados que dirigen departamentos en la ubicación especificada por el usuario. Se puede suponer que ninguno dirige más de un departamento.
1. Describirse las estructuras de archivos y los índices que se escogerían.
 2. Posteriormente se hace evidente que las actualizaciones de estas relaciones son frecuentes. Como los índices suponen una gran sobrecarga, ¿se puede pensar en un modo de mejorar el rendimiento de estas consultas sin emplear índices?

Ejercicio 13.8 Para cada una de las consultas siguientes, identifíquese un posible motivo por el que un optimizador no podría encontrar un buen plan. Reescríbase la consulta de modo que sea probable hallar un buen plan. Los índices disponibles o las restricciones conocidas se muestran delante de cada consulta; supóngase que los esquemas de las relaciones son consistentes con los atributos a los que se hace referencia en la consulta.

1. Se dispone de un índice para el atributo *edad*:

```
SELECT E.númd
FROM Empleado E
WHERE E.edad=20 OR E.edad=10
```

2. Se dispone de un índice de árbol B+ para el atributo *edad*:

```
SELECT E.númd
FROM Empleado E
WHERE E.edad<20 AND E.edad>10
```

3. Se dispone de un índice para el atributo *edad*:

```
SELECT E.númd
FROM Empleado E
WHERE 2*E.edad<20
```

4. No se dispone de ningún índice:

```
SELECT DISTINCT *
FROM Empleado E
```

5. No se dispone de ningún índice:

```
SELECT AVG (E.sueldo)
FROM Empleado E
GROUP BY E.númd
HAVING E.númd=22
```

6. El atributo *idm* de Reservas es una clave externa que hace referencia a Marineros:

```
SELECT M.idm
FROM Marineros M, Reservas R
WHERE M.idm=R.idm
```

Ejercicio 13.9 Considérense dos maneras de calcular el nombre de los empleados que ganan más de 100.000 € y cuya edad es igual a la de su responsable. En primer lugar, una consulta anidada:

```
SELECT E1.nombrree
FROM Emp E1
WHERE E1.sueldo > 100 AND E1.edad = ( SELECT E2.edad
                                         FROM Emp E2, Dep D2
                                         WHERE E1.nombrred = D2.nombrred
                                               AND D2.resp = E2.nombrree )
```

En segundo lugar, una consulta que emplea la definición de una vista:

```
SELECT E1.nombrree
FROM Emp E1, EdadResp A
WHERE E1.nombrred = A.nombrred AND E1.sueldo > 100 AND E1.edad = A.edad

CREATE VIEW EdadResp (nombrred, edad)
AS SELECT D.nombrred, E.edad
        FROM Emp E, Dep D
        WHERE D.resp = E.nombrree
```

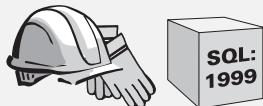
1. Describese una situación en la que sea probable que la primera consulta supere en rendimiento a la segunda.
2. Describese una situación en la que sea probable que la segunda consulta supere en rendimiento a la primera.
3. ¿Se puede crear una consulta equivalente que sea probable que supere a estas dos consultas cuando todos los empleados que ganan más de 100.000 € tengan 35 o 40 años de edad? Explíquese brevemente.

NOTAS BIBLIOGRÁFICAS

Shasha y Bonnet dan una amplia exhaustiva introducción al ajuste de bases de datos [61]. [394] es un análisis pionero del diseño físico de bases de datos. [395] analiza las implicaciones de la normalización en el rendimiento y observa que la desnormalización puede mejorar el rendimiento de ciertas consultas. Las ideas subyacentes a una herramienta de IBM para el diseño físico se describen en [180]. La herramienta AutoAdmin de Microsoft que lleva a cabo la selección automática de índices de acuerdo con la carga de trabajo de cada consulta se describe en varios trabajos [102, 103]. El Asesor de DB2 se describe en [458]. Otros enfoques del diseño físico de bases de datos se describen en [90, 383]. [407] considera el *ajuste de transacciones*, que aquí sólo se ha analizado brevemente. El problema es cómo estructurar una aplicación en un conjunto de transacciones para maximizar el rendimiento.

Los siguientes libros sobre diseño de bases de datos tratan detalladamente problemas de diseño físico; se recomiendan para posteriores lecturas. [182] es bastante independiente de los productos concretos, aunque muchos de los ejemplos se basan en DB2 y en los sistemas de Teradata. [474] trata principalmente de DB2. [406] es un tratamiento muy fácil de leer del ajuste del rendimiento y no es específico de ningún sistema concreto.

[217] contiene varios trabajos sobre las pruebas de rendimiento de los sistemas de bases de datos y tiene software adjunto. Incluye artículos sobre las pruebas de rendimiento *AS³AP*, Set Query, TPC-A, TPC-B, Wisconsin y 001 escritos por sus desarrolladores originales. La prueba de rendimiento Bucky se describe en [80], la prueba de rendimiento 007 se describe en [79] y la prueba de rendimiento TPC-D se describe en [448]. La prueba de rendimiento Sequoia 2000 se describe en [437].



14

SEGURIDAD Y AUTORIZACIÓN

- ➔ ¿Cuáles son las principales consideraciones de seguridad al diseñar aplicaciones de bases de datos?
- ➔ ¿Qué mecanismos ofrecen los SGBD para controlar el acceso de los usuarios a los datos?
- ➔ ¿Qué es el control discrecional de acceso y cómo lo soporta SQL?
- ➔ ¿Cuáles son las debilidades del control discrecional de acceso? ¿Cómo se abordan en el control obligatorio de acceso?
- ➔ ¿Qué son los canales clandestinos y cómo ponen en peligro el control obligatorio de acceso?
- ➔ ¿Qué debe hacer el DBA para garantizar la seguridad?
- ➔ ¿Cuál es la amenaza de seguridad añadida cuando se tiene acceso de forma remota a una base de datos?
- ➔ ¿Cuál es el papel del cifrado a la hora de garantizar un acceso seguro? ¿Cómo se utiliza para certificar servidores y crear firmas digitales?
- ➡ **Conceptos fundamentales:** seguridad, integridad, disponibilidad; control discrecional de acceso, privilegios, GRANT, REVOKE; control obligatorio de acceso, objetos, sujetos, clases de seguridad, tablas multinivel, poliinstanciación; canales clandestinos, niveles de seguridad del DoD; bases de datos estadísticas, inferencia de información segura; autenticación para el acceso remoto, protección de servidores, firmas digitales; cifrado, cifrado de clave pública.

Sé que es un secreto porque se susurra por todas partes.

— William Congreve

Los datos guardados en un SGBD suelen ser vitales para los intereses empresariales de la organización y se consideran un activo corporativo. Además de proteger el valor intrínseco de

esos datos, las empresas deben considerar la manera de garantizar la intimidad y de controlar el acceso a los datos que no se deban revelar a determinados grupos de usuarios por diferentes razones.

En este capítulo se analizan los conceptos subyacentes al control de acceso y a la seguridad de los SGBD. Tras introducir los aspectos de seguridad de las bases de datos en el Apartado 14.1, se considerarán dos enfoques distintos, denominados *discrecional* y *obligatorio*, para especificar y gestionar los controles de acceso. Los mecanismos de **control de acceso** son una manera de controlar los datos accesibles para cada usuario. Tras introducir los controles de acceso en el Apartado 14.2, se tratará el control discrecional de acceso, que está soportado en SQL, en el Apartado 14.3. Se tratará brevemente el control obligatorio de acceso, que no está soportado en SQL, en el Apartado 14.4.

En el Apartado 14.6 se analizan algunos aspectos adicionales de la seguridad de las bases de datos, como la seguridad en las bases de datos estadísticas y el papel del administrador de la base de datos. A continuación se consideran en el Apartado 14.5 algunos de los desafíos peculiares del soporte de acceso seguro a los SGBD a través de Internet, que es un problema primordial del comercio electrónico y de otras aplicaciones de las bases de datos en Internet. Se concluye este capítulo con un análisis de los aspectos de seguridad del caso de estudio de Benito y Norberto en el Apartado 14.7.

14.1 INTRODUCCIÓN A LA SEGURIDAD DE LAS BASES DE DATOS

Hay tres objetivos principales cuando se diseña una aplicación de bases de datos segura:

1. **Secreto.** La información no se debe dar a conocer a usuarios no autorizados. Por ejemplo, no se debe permitir que un alumno examine las notas de otros.
2. **Integridad.** Sólo se debe permitir modificar los datos a los usuarios autorizados. Por ejemplo, puede que se permita que los alumnos vean sus notas, pero (evidentemente) no que las modifiquen.
3. **Disponibilidad.** No se debe impedir el acceso a los usuarios autorizados. Por ejemplo, se debe permitir que un profesor que lo deseé pueda cambiar una nota.

Para alcanzar estos objetivos hay que desarrollar una **política de seguridad** clara y consistente que describa las medidas de seguridad que se deban aplicar. En concreto hay que determinar la parte de los datos que se debe proteger y los usuarios que tendrán acceso a cada parte de los datos. A continuación hay que utilizar el **mecanismo de seguridad** del SGBD y del sistema operativo subyacentes, así como mecanismos externos como la protección de los accesos a los edificios, para hacer que se cumpla esa política. Hay que destacar que se deben tomar medidas de seguridad a varios niveles.

Los fallos de seguridad del SO o de las conexiones de red pueden eludir los mecanismos de seguridad de la base de datos. Por ejemplo, esos fallos pueden permitir que un intruso inicie sesión como administrador de la base de datos, con todos los derechos de acceso al SGBD consiguientes. El factor humano es otra fuente de fallos de seguridad. Por ejemplo, puede que un usuario escoja una contraseña que sea fácil de adivinar o que un usuario autorizado a ver

datos delicados les dé un mal uso. Estos errores suponen un gran porcentaje de los fallos de seguridad. Estos aspectos de la seguridad no se discutirán aquí pese a su importancia, ya que no son específicos de los sistemas de gestión de bases de datos; nuestra atención se centra en los mecanismos de control del acceso a las bases de datos que soportan las políticas de seguridad.

Las vistas son una herramienta valiosa para el cumplimiento de las políticas de seguridad. Se puede emplear el mecanismo de las vistas para crear “ventanas” a conjuntos de datos que resulten adecuadas para grupos de usuarios determinados. Las vistas nos permiten limitar el acceso a datos delicados ofreciendo acceso a una versión restringida (definida mediante una vista) de esos datos, más que a los propios datos.

En los ejemplos se emplearán los esquemas siguientes:

```
Marineros(idm: integer, nombrem: string, categoría: integer, edad: real)
Barcos(idb: integer, nombreb: string, color: string)
Reservas(idm: integer, idb: integer, fecha: dates)
```

Cada vez más, a medida que los sistemas de bases de datos se van convirtiendo en el sostén principal de las aplicaciones de comercio electrónico, las solicitudes provienen de Internet. Esto hace que sea importante poder **autenticar** a los usuarios ante el sistema de bases de datos. Después de todo, hacer que se cumpla una política de seguridad que permite que el usuario Samuel lea una tabla y el usuario Enrique la escriba no resulta muy útil si Samuel se puede hacer pasar por Enrique. A la inversa, se debe poder garantizar a los usuarios que se están comunicando con el sistema auténtico (por ejemplo, el verdadero servidor de Amazon.com, y no una aplicación espuria que pretende robar información delicada como el número de la tarjeta de crédito). Aunque los detalles de la autenticación caen fuera de los objetivos de este libro, se analizan el papel de la autenticación y sus principios básicos en el Apartado 14.5, una vez tratados los mecanismos de control del acceso a las bases de datos.

14.2 CONTROL DE ACCESO

Las bases de datos de las empresas contienen gran cantidad de información y suelen tener varios grupos de usuarios. La mayor parte de los usuarios sólo necesita tener acceso a una pequeña parte de la base de datos para llevar a cabo sus tareas. Puede no ser deseable que se permita a los usuarios el acceso sin restricciones a la totalidad de los datos, y el SGBD debe ofrecer mecanismos para controlar el acceso a los datos.

Los SGBD ofrecen dos enfoques principales del control de acceso. El **control discrecional de acceso** se basa en el concepto de derecho de acceso, o **privilegio**, y en los mecanismos para conceder a los usuarios esos privilegios. Los privilegios permiten a los usuarios el acceso a determinados objetos de datos de cierta manera (por ejemplo, para leerlos o para modificarlos). El usuario que crea un objeto de la base de datos como una tabla o una vista consigue automáticamente todos los privilegios aplicables a ese objeto. El SGBD realiza posteriormente un seguimiento del modo en que se conceden, y quizás se revoquen, esos privilegios a otros usuarios y garantiza que en todo momento sólo puedan tener acceso a ese objeto los usuarios con los privilegios necesarios. SQL soporta el control discrecional de acceso mediante las órdenes **GRANT** y **REVOKE**. La orden **GRANT** concede privilegios a los usuarios, mientras que la orden **REVOKE** se los retira. El control discrecional de acceso se analiza en el Apartado 14.3.

Los mecanismos de control discrecional de acceso, aunque suelen ser efectivos, tienen algunos puntos débiles. En concreto, un usuario no autorizado con malas intenciones puede engañar a un usuario autorizado para que le revele datos delicados. El **control obligatorio de acceso** se basa en políticas que se aplican a todo el sistema y que los usuarios no pueden modificar. En este enfoque se le asigna a cada objeto de la base de datos una *clase de seguridad*, a cada usuario se le asigna una *autorización* para una clase de seguridad y se imponen reglas a los usuarios sobre la lectura y la escritura de los objetos de la base de datos. El SGBD determina si un usuario determinado puede leer o escribir un objeto dado basándose en ciertas reglas que tienen que ver con el nivel de seguridad del objeto y la autorización del usuario. Esas reglas pretenden garantizar que nunca se puedan “pasar” datos delicados a usuarios que no tengan la autorización correspondiente. La norma de SQL no incluye el soporte del control obligatorio de acceso. El control obligatorio de acceso se analiza en el Apartado 14.4.

14.3 CONTROL DISCRECIONAL DE ACCESO

SQL soporta el control discrecional de acceso mediante las órdenes GRANT y REVOKE. La orden GRANT concede a los usuarios privilegios sobre las tablas básicas y las vistas. La sintaxis de esta orden es la siguiente:

```
GRANT privilegios ON objeto TO usuarios [ WITH GRANT OPTION ]
```

Para lo que nos interesa, un **objeto** es una tabla básica o una vista. SQL reconoce otros tipos de objetos, pero no se analizarán aquí. Se pueden especificar varios privilegios, incluidos los siguientes:

- **SELECT.** El derecho a tener acceso (leer) todas las columnas de la tabla especificada como **objeto**, *incluidas las columnas añadidas posteriormente* mediante órdenes ALTER TABLE.
- **INSERT(*nombre-columna*).** El derecho a insertar filas con valores (no-null o no predeterminados) en la columna indicada de la tabla señalada como **objeto**. Si hay que conceder este derecho con respecto a todas las columnas, incluidas las columnas que se puedan añadir posteriormente, se puede emplear simplemente INSERT. Los privilegios UPDATE(*nombre-columna*) y UPDATE son parecidos.
- **DELETE.** El derecho a eliminar filas de la tabla señalada como **objeto**.
- **REFERENCES(*nombre-columna*).** El derecho a definir claves externas (de otras tablas) que hagan referencia a la columna especificada de la tabla **objeto**. REFERENCES sin nombre de columna especificado denota este derecho con respecto a todas las columnas, incluidas las que se añadan posteriormente.

Si un usuario tiene un privilegio con la **opción grant**, puede transmitírselo a otro usuario (con la opción grant o sin ella) mediante la orden GRANT. Los usuarios que crean tablas básicas tienen automáticamente todos los privilegios aplicables sobre esas tablas, junto con el derecho a concedérselos a otros usuarios. Los usuarios que crean vistas tienen exactamente los mismos privilegios sobre esas vistas que tengan sobre *todas* las vistas o tablas básicas empleadas para definir esas vistas. Por supuesto, los usuarios que crean vistas deben tener el privilegio SELECT

Autorización basada en los roles de SQL. En SQL-92 los privilegios se asignan a los usuarios (a los identificadores de autorización, para ser precisos). En el mundo real los privilegios suelen estar asociados al trabajo del usuario o a su *rol* dentro de la organización. Muchos SGBD llevan tiempo dando soporte al concepto de **rol** y permitiendo que los privilegios se asignen a los roles. Por tanto, los roles se pueden conceder a los usuarios y a otros roles. (Por supuesto, los privilegios también se pueden conceder directamente a los usuarios.) La norma SQL:1999 incluye el soporte de los roles. Los roles se pueden crear y destruir mediante las órdenes **CREATE ROLE** y **DROP ROLE**. Se pueden conceder roles a los usuarios (opcionalmente, con la posibilidad de transmitir el rol a otros). Las órdenes **GRANT** y **REVOKE** estándar pueden asignar privilegios a roles o a identificadores de autorización (y revocárselos).

¿Cuál es la ventaja de incluir una característica que ya soportan muchos sistemas? Esto garantiza que, con el tiempo, todos los fabricantes que cumplan con la norma sopor tarán esta característica. Por tanto, los usuarios pueden utilizar esta característica sin preocuparse por la portabilidad de sus aplicaciones entre diferentes SGBD.

sobre todas las tablas subyacentes y, por tanto, siempre se concede el privilegio **SELECT** sobre la vista. El creador de la vista sólo tiene el privilegio **SELECT** con la opción conceder si tiene el privilegio **SELECT** con la opción conceder sobre todas las tablas subyacentes. Además, si la vista es actualizable y el usuario tiene los privilegios **INSERT**, **DELETE** o **UPDATE** (con la opción grant o sin ella) sobre la (única) tabla subyacente, el usuario obtiene automáticamente los mismos privilegios sobre la vista.

Sólo el propietario del esquema puede ejecutar las instrucciones de definición de datos **CREATE**, **ALTER** y **DROP** en ese esquema. El derecho a ejecutar esas instrucciones no se puede conceder ni revocar.

Junto con las órdenes **GRANT** y **REVOKE**, las vistas son un componente importante de los mecanismos de seguridad proporcionados por los SGBD relacionales. Mediante la definición de vistas sobre las tablas básicas se puede presentar a los usuarios la información que necesitan mientras se les *oculta* otra información a la que no se les debe dar acceso. Por ejemplo, considérese la siguiente definición de vista:

```
CREATE VIEW MarinerosActivos (nombre, edad, fecha)
    AS SELECT M.nombrem, M.edad, R.fecha
        FROM   Marineros M, Reservas R
        WHERE  M.idm = R.idm AND S.categoría > 6
```

Los usuarios que tengan acceso a **MarinerosActivos**, pero no a **Marineros** ni a **Reservas**, conoce el nombre de los marineros que han realizado reservas, pero no puede averiguar el *idb* de los barcos reservados por cada marinero.

En SQL los privilegios se asignan a los **identificadores de autorizaciones**, que pueden denotar a un solo usuario o a un grupo de usuarios; el usuario debe especificar un identificador de autorización y, en muchos sistemas, la *contraseña* correspondiente antes de que el SGBD acepte ninguna orden. Por tanto, en los ejemplos siguientes, *José*, *Miguel* y el resto son técnicamente identificadores de autorizaciones en vez de nombres de usuario.

Supóngase que el usuario José ha creado las tablas Barcos, Reservas y Marineros. A continuación pueden verse algunos ejemplos de la orden GRANT que ahora puede ejecutar José:

```
GRANT INSERT, DELETE ON Reservas TO Yago WITH GRANT OPTION
GRANT SELECT ON Reservas TO Miguel
GRANT SELECT ON Marineros TO Miguel WITH GRANT OPTION
GRANT UPDATE (categoría) ON Marineros TO Luisa
GRANT REFERENCES (idb) ON Barcos TO Bonifacio
```

Yago puede insertar o eliminar filas de Reservas y autorizar a otros a hacer lo mismo. Miguel puede ejecutar consultas SELECT sobre Marineros y sobre Reservas y puede transmitir ese privilegio a otros para Marineros, pero no para Reservas. Con el privilegio SELECT, Miguel puede crear vistas que tengan acceso a las tablas Marineros y Reservas (por ejemplo, la vista MarinerosActivos) pero no puede conceder a otros el privilegio SELECT sobre MarinerosActivos.

Por otro lado, supóngase que Miguel crea la vista siguiente:

```
CREATE VIEW MarinerosJóvenes (idm, edad, categoría)
AS SELECT M.idm, M.edad, M.categoría
FROM Marineros M
WHERE M.edad < 18
```

La única tabla subyacente es Marineros, para la cual Miguel tiene el privilegio SELECT con la opción grant. Por tanto, tiene el privilegio SELECT con la opción grant sobre MarinerosJóvenes y puede transmitir el privilegio SELECT sobre MarinerosJóvenes a Enrique y a Gonzalo:

```
GRANT SELECT ON MarinerosJóvenes TO Enrique, Gonzalo
```

Enrique y Gonzalo pueden ejecutar ahora consultas SELECT sobre la vista MarinerosJóvenes —obsérvese, no obstante, que Enrique y Gonzalo *no* tienen derecho a ejecutar consultas SELECT directamente sobre la tabla Marineros subyacente—.

Miguel puede definir también restricciones basadas en la información de las tablas Marineros y Reservas. Por ejemplo, Miguel puede definir la tabla siguiente, que tiene una restricción de tabla asociada:

```
CREATE TABLE Furtiva (categoríamax INTEGER,
CHECK ( categoríamax >=
        ( SELECT MAX (M.categoría )
          FROM Marineros M )) )
```

Mediante la inserción reiterada de filas en la tabla Furtiva con valores de *categoríamax* que se incrementan gradualmente hasta que una inserción finalmente tiene éxito, Miguel puede averiguar el valor máximo de *categoría* en la tabla Marineros. Este ejemplo ilustra el motivo de que SQL exija que los creadores de restricciones de tabla que hagan referencia a Marineros posean el privilegio SELECT sobre Marineros.

Volviendo a los privilegios concedidos por José, Luisa sólo puede actualizar la columna *categoría* de las filas de Marineros. Puede ejecutar la siguiente orden, que asigna el valor de 8 a todas las categorías:

```
UPDATE Marineros M
SET     M.categoría = 8
```

Sin embargo, no puede ejecutar el mismo comando si se cambia la cláusula SET por SET *M.edad = 25*, ya que no está autorizada a actualizar el campo *edad*. Un detalle más sutil queda ilustrado por la siguiente orden, que disminuye la categoría de todos los marineros:

```
UPDATE Marineros M
SET     M.categoría = M.categoría - 1
```

Luisa no puede ejecutar esta orden porque exige el privilegio SELECT sobre la columna *M.categoría*, y ella no lo tiene.

Bonifacio puede hacer referencia a la columna *edb* de Barcos como clave externa de otra tabla. Por ejemplo, puede crear la tabla Reservas mediante la orden siguiente:

```
CREATE TABLE Reservas (
    idm    INTEGER,
    idb    INTEGER,
    fecha  DATE,
    PRIMARY KEY (idb, fecha),
    FOREIGN KEY (idm) REFERENCES Marineros ,
    FOREIGN KEY (idb) REFERENCES Barcos )
```

Si Bonifacio no tuviera el privilegio REFERENCES sobre la columna *edb* de Barcos, no podría ejecutar esta instrucción CREATE, ya que la cláusula FOREIGN KEY exige este privilegio. (Algo parecido se cumple con respecto a la referencia de la clave externa a Marineros.)

Especificar sólo el privilegio INSERT (análogamente, REFERENCES y otros privilegios) en una orden GRANT no es igual que especificar SELECT(*nombre-columna*) para cada columna presente en la tabla. Considérese la siguiente orden sobre la tabla Marineros, que tiene las columnas *idm*, *nombrem*, *categoría* y *edad*:

```
GRANT INSERT ON Marineros TO Miguel
```

Supóngase que se ejecuta esta orden y luego se añade una columna a la tabla Marineros (mediante la ejecución de una orden ALTER TABLE). Obsérvese que Miguel tenía el privilegio INSERT con respecto a la columna recién añadida. Si se hubiera ejecutado la siguiente orden GRANT, en lugar de la anterior, Miguel no tendría el privilegio INSERT sobre la columna nueva:

```
GRANT INSERT ON Marineros(idm), Marineros(nombrem), Marineros(categoría),
    Marineros(edad), TO Miguel
```

Hay una orden complementaria a GRANT que permite la retirada de los privilegios. La sintaxis de la orden REVOKE es la siguiente:

```
REVOKE [ GRANT OPTION FOR ] privilegios
    ON objeto FROM usuarios { RESTRICT | CASCADE }
```

Esta orden se puede utilizar para revocar privilegios o sólo la opción grant de esos privilegios (mediante la cláusula opcional GRANT OPTION FOR). Hay que especificar una de las dos alternativas, RESTRICT o CASCADE; puede verse fácilmente lo que significa esta opción.

La intuición subyacente a la orden **GRANT** está clara: se concede a los creadores de tablas básicas o de vistas todos los privilegios correspondientes con respecto a esa tabla o vista y se le permite transmitir esos privilegios —incluido el derecho a transmitir privilegios— a otros usuarios. La orden **REVOKE** está pensada, como cabía esperar, para conseguir lo contrario: puede que el usuario que ha concedido un privilegio a otro usuario cambie de opinión y desee retirar el privilegio concedido. La intuición subyacente al efecto que tiene la orden **REVOKE** se complica por el hecho de que se le puede conceder a cada usuario el mismo privilegio varias veces, posiblemente otorgado por diferentes usuarios.

Cuando el usuario ejecuta la orden **REVOKE** con la palabra clave **CASCADE**, el efecto es la retirada de los privilegios indicados o la opción **grant** a todos los usuarios que tengan esos privilegios en ese momento *exclusivamente* debido a una orden **GRANT** que el mismo usuario que ahora ejecuta la orden **REVOKE** haya ejecutado previamente. Si esos usuarios recibieron los privilegios con la opción **grant** y los transmitieron, los receptores perderán a su vez esos privilegios como consecuencia de la orden **REVOKE**, a menos que los recibieran mediante otra orden **GRANT** diferente.

La orden **REVOKE** se ilustrará mediante varios ejemplos. En primer lugar, considérese lo que ocurre tras la siguiente secuencia de órdenes, donde José es el creador de *Marineros*.

GRANT SELECT ON Marineros TO Arturo WITH GRANT OPTION	<i>(ejecutada por José)</i>
GRANT SELECT ON Marineros TO Borja WITH GRANT OPTION	<i>(ejecutada por Arturo)</i>
REVOKE SELECT ON Marineros FROM Arturo CASCADE	<i>(ejecutada por José)</i>

Arturo pierde el privilegio **SELECT** sobre *Marineros*, por supuesto. Luego Borja, que recibió ese privilegio de Arturo, y sólo de Arturo, también lo pierde. Se dice que el privilegio de Borja se **abandona** cuando se revoca el privilegio del que se derivaba (el privilegio **SELECT** con la opción **grant** de Arturo, en este ejemplo). Cuando se especifica la palabra clave **CASCADE**, también se revocan todos los privilegios abandonados (lo que quizás haga que los privilegios de otros usuarios queden abandonados y, por tanto, se revoquen de manera recursiva). Si se especifica la palabra **RESTRICT** en la orden **REVOKE**, ésta se rechaza si la revocación de los privilegios *exclusivamente* de los usuarios especificados en la orden da lugar a que queden abandonados otros privilegios.

Considérese la secuencia siguiente como ejemplo adicional:

GRANT SELECT ON Marineros TO Arturo WITH GRANT OPTION	<i>(ejecutada por José)</i>
GRANT SELECT ON Marineros TO Borja WITH GRANT OPTION	<i>(ejecutada por José)</i>
GRANT SELECT ON Marineros TO Borja WITH GRANT OPTION	<i>(ejecutada por Arturo)</i>
REVOKE SELECT ON Marineros FROM Arturo CASCADE	<i>(ejecutada por José)</i>

Como antes, Arturo pierde el privilegio **SELECT** sobre *Marineros*. ¿Pero qué pasa con Borja? Borja recibió ese privilegio de Arturo, pero también lo ha recibido de manera independiente (casualmente, directamente de José). Por tanto, Borja conserva el privilegio. Considérese un tercer ejemplo:

GRANT SELECT ON Marineros TO Arturo WITH GRANT OPTION	<i>(ejecutada por José)</i>
GRANT SELECT ON Marineros TO Arturo WITH GRANT OPTION	<i>(ejecutada por José)</i>
REVOKE SELECT ON Marineros FROM Arturo CASCADE	<i>(ejecutada por José)</i>

Dado que José concedió el privilegio a Arturo dos veces y sólo lo ha revocado una, ¿conserva Arturo ese privilegio? En lo que respecta a la norma de SQL, no. Aunque José concedió por despiste varias veces el mismo privilegio a Arturo, lo puede revocar con una sola orden REVOKE.

Se puede revocar sólo la opción grant del privilegio:

```
GRANT SELECT ON Marineros TO Arturo WITH GRANT OPTION      (ejecutada por José)
REVOKE GRANT OPTION FOR SELECT ON Marineros
FROM Arturo CASCADE                                         (ejecutada por José)
```

Esta orden dejaría a Arturo con el privilegio SELECT sobre Marineros, pero ya no tendría la opción grant para ese privilegio y, por tanto, no podría transmitirlo a otros usuarios.

Estos ejemplos sacan a la luz la intuición subyacente a la orden REVOKE y destacan la compleja interacción entre las órdenes GRANT y REVOKE. Cuando se ejecuta una orden GRANT, se añade un **descriptor de privilegios** a una tabla de descriptores de este tipo que mantiene el SGBD. El descriptor de privilegios especifica lo siguiente: el *adjudicador* del privilegio, el *adjudicatario* del privilegio, el *privilegio adjudicado* (incluido el nombre del objeto implicado) y si se incluye la opción grant. Cuando un usuario crea una tabla o una vista y obtiene determinados privilegios “automáticamente”, se añade a esa tabla un descriptor de privilegios con el *sistema* como adjudicador.

El efecto de una serie de órdenes GRANT se puede describir en términos de un **grafo de autorizaciones** en el que los nodos son los usuarios —técnicamente son identificadores de autorizaciones— y los arcos indican la manera en que se transmiten los privilegios. Hay un arco desde el (nodo para el) usuario 1 al usuario 2 si el usuario 1 ha ejecutado alguna orden GRANT para conceder un privilegio al usuario 2; el arco se etiqueta con el descriptor para la orden GRANT. La orden GRANT no tiene ningún efecto si ya se han concedido los mismos privilegios al mismo adjudicatario por el mismo adjudicador. La siguiente secuencia de órdenes ilustra la semántica de las órdenes GRANT y REVOKE cuando hay un *ciclo* en el grafo de autorizaciones:

```
GRANT SELECT ON Marineros TO Arturo WITH GRANT OPTION      (ejecutada por José)
GRANT SELECT ON Marineros TO Borja WITH GRANT OPTION     (ejecutada por Arturo)
GRANT SELECT ON Marineros TO Arturo WITH GRANT OPTION    (ejecutada por Borja)
GRANT SELECT ON Marineros TO Carlos WITH GRANT OPTION   (ejecutada por José)
GRANT SELECT ON Marineros TO Borja WITH GRANT OPTION    (ejecutada por Carlos)
REVOKE SELECT ON Marineros FROM Arturo CASCADE           (ejecutada por José)
```

El grafo de autorizaciones para este ejemplo puede verse en la Figura 14.1. Obsérvese que se indica la manera en que José, el creador de Marineros, consiguió el privilegio SELECT del SGBD mediante la introducción del nodo *Sistema* y el trazado de un arco desde ese nodo al de José.

Como el grafo indica claramente, la adjudicación de Borja a Arturo y la de Arturo a Borja (del mismo privilegio) crean un ciclo. Posteriormente Carlos le concede el mismo privilegio a Borja, que lo había recibido de José de manera independiente. En ese momento José decide revocar el privilegio que había concedido a Arturo.

Sigamos el efecto de esta revocación. El arco de José a Arturo se elimina porque corresponde a la acción de adjudicación que se ha revocado. Todos los nodos restantes tienen la propiedad siguiente: *si el nodo N tiene un arco saliente etiquetado con un privilegio, hay un*

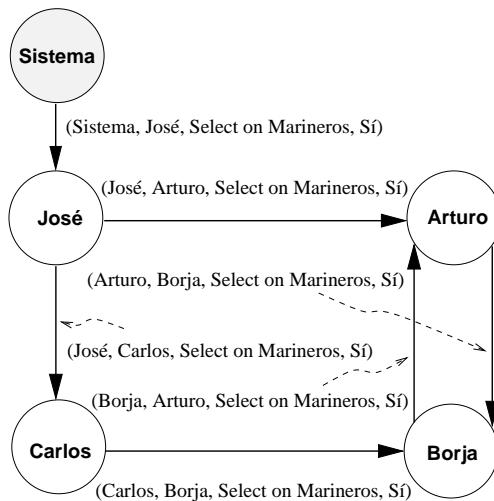


Figura 14.1 Ejemplo de grafo de autorizaciones

camino desde el nodo Sistema al nodo N en el que cada etiqueta de arco contiene el mismo privilegio más la opción grant. Es decir, cualquier acción de adjudicación restante queda justificada por un privilegio recibido (directa o indirectamente) del Sistema. La ejecución de la orden REVOKE de José, por tanto, se detiene en este punto, en el que todo el mundo conserva el privilegio SELECT sobre Marineros.

Puede parecer que este resultado va contra la intuición, ya que Arturo sigue teniendo el privilegio sólo porque lo recibió de Borja y, en el momento en que Borja concedió el privilegio a Arturo, sólo lo había recibido de Arturo. Aunque Borja consiguió posteriormente el privilegio de Carlos, ¿no se debería deshacer el efecto de su adjudicación a Arturo al ejecutar la orden REVOKE de José? En SQL el efecto de la adjudicación de Borja a Arturo *no* se deshace. En efecto, si un usuario consigue un privilegio varias veces de adjudicadores diferentes, SQL trata a cada una de esas adjudicaciones como si hubieran ocurrido *antes* de que el usuario transmitiera el privilegio a otros usuarios. Esta implementación de REVOKE resulta conveniente en muchas situaciones reales. Por ejemplo, si se despide a un administrador después de que haya transmitido algún privilegio a sus subordinados (que, a su vez, pueden habérselos transmitido a otros), se puede garantizar que sólo se eliminan los privilegios del administrador si se vuelven a llevar a cabo todas las acciones de adjudicación del administrador y luego se revocan sus privilegios. Es decir, no hace falta rehacer de manera recursiva las acciones de adjudicación de los subordinados.

Para volver a la saga de José y de sus amigos, supóngase que José decide revocar también el privilegio SELECT de Carlos. Evidentemente, se elimina el arco de José a Carlos correspondiente a la adjudicación de este privilegio. También se elimina el arco de Carlos a Borja, puesto que ya no hay ningún camino de Sistema a Carlos que conceda a Carlos el derecho de conceder a Borja el privilegio SELECT sobre Marineros. El grafo de autorizaciones en este punto intermedio puede verse en la Figura 14.2.

El grafo contiene ahora dos nodos (Arturo y Borja) para los que hay arcos salientes con etiquetas que contienen el privilegio SELECT sobre Marineros; por tanto, esos usuarios han

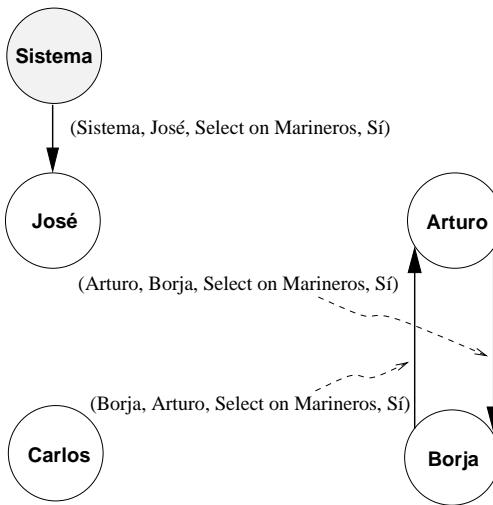


Figura 14.2 Ejemplo de grafo de autorizaciones durante la revocación

concedido ese privilegio. Sin embargo, aunque cada nodo contiene un arco entrante que lleva el mismo privilegio, *no existe camino desde Sistema a ninguno de esos nodos*; por lo tanto, el derecho de esos usuarios a conceder el privilegio se ha abandonado. Por tanto, también se eliminan los arcos salientes. En general, puede que esos nodos tengan otros arcos que incidan sobre ellos pero, en este ejemplo, ahora no tienen ningún arco que incida sobre ellos. José queda como el único usuario con el privilegio SELECT sobre Marineros; Arturo y Borja han perdido sus privilegios.

14.3.1 Concesión y revocación de vistas y restricciones de integridad

Los privilegios que tiene el creador de una vista (con respecto a esa vista) cambian con el tiempo a medida que gana o pierde privilegios sobre las tablas subyacentes. Si el creador pierde algún privilegio que tuviera con la opción grant, también lo pierden los usuarios a los que se les concedió ese privilegio sobre la vista. Hay algunos aspectos sutiles de las órdenes GRANT y REVOKE cuando tienen que ver con vistas o con restricciones de integridad. Se tomarán en consideración varios ejemplos que destacarán los siguientes puntos de importancia:

1. Se puede descartar una vista porque se haya revocado el privilegio SELECT al usuario que la creó.
2. Si el creador de una vista obtiene privilegios adicionales sobre las tablas subyacentes, también obtiene de manera automática privilegios adicionales sobre esa vista.
3. La distinción entre los privilegios REFERENCES y SELECT es importante.

Supóngase que José creó Marineros y concedió a Miguel el privilegio SELECT sobre esa tabla con la opción grant y que Miguel creó luego la vista MarinerosJóvenes y concedió a Enrique

el privilegio **SELECT** sobre *MarinerosJóvenes*. Enrique define ahora una vista denominada *BuenosMarinerosJóvenes*:

```
CREATE VIEW BuenosMarinerosJóvenes (nombre, edad, categoría)
AS SELECT M.nombrem, M.edad, M.categoría
FROM   MarinerosJóvenes M
WHERE  M.categoría > 6
```

¿Qué pasa si José revoca a Miguel el privilegio **SELECT** sobre *Marineros*? Miguel ya no tiene permiso para ejecutar la consulta empleada para definir *MarinerosJóvenes*, ya que la definición hace referencia a *Marineros*. Por tanto, se descarta (es decir, se destruye) la vista *MarinerosJóvenes*. A su vez, también se descarta *BuenosMarinerosJóvenes*. Las definiciones de las dos vistas se eliminan de los catálogos del sistema; aunque un José arrepentido decidiera devolverle el privilegio **SELECT** sobre *Marineros* a Miguel, las vistas ya no existen y habría que crearlas de nuevo si hicieran falta.

Como añadido, supóngase que todo tiene lugar como se acaba de describir hasta que Enrique define *BuenosMarinerosJóvenes*; entonces, en lugar de revocar a Miguel el privilegio **SELECT** sobre *Marineros*, José decide concederle también el privilegio **INSERT** sobre *Marineros*. Los privilegios de Miguel sobre la vista *MarinerosJóvenes* se equiparan a los que tendría si fuera a crear la vista *ahora*. Por tanto, también consigue el privilegio **INSERT** sobre *MarinerosJóvenes*. (Obsérvese que esa vista es actualizable.) ¿Qué pasa con Enrique? Sus privilegios no se modifican.

Que Miguel tenga o no el privilegio **INSERT** con la opción *grant* sobre *MarinerosJóvenes* depende de que José le conceda el privilegio **INSERT** sobre *Marineros* con la opción *grant*. Para comprender esta situación, considérese otra vez la situación de Enrique. Si Miguel tiene el privilegio **INSERT** con la opción *grant* sobre *MarinerosJóvenes*, puede transmitírselo a Enrique. Entonces, Enrique podría insertar filas en la tabla *Marineros*, ya que las inserciones en *MarinerosJóvenes* se llevan a cabo modificando la tabla básica subyacente, *Marineros*. Evidentemente, no se desea que Miguel pueda realizar esas modificaciones a menos que tenga el privilegio **INSERT** con la opción *grant* sobre *Marineros*.

El privilegio **REFERENCES** es muy diferente del privilegio **SELECT**, como ilustra el ejemplo siguiente. Supóngase que José es el creador de *Barcos*. Puede autorizar que otro usuario, por ejemplo, Francisco, cree *Reservas* con una clave externa que haga referencia a la columna *idb* de *Barcos* concediéndole el privilegio **REFERENCES** con respecto a esa columna. Por otro lado, si Francisco tiene el privilegio **SELECT** sobre la columna *idb* de *Barcos* pero no el privilegio **REFERENCES**, *no puede* crear *Reservas* con una clave externa que haga referencia a *Barcos*. Si Francisco crea *Reservas* con una clave externa que haga referencia a *idb* de *Barcos* y luego pierde el privilegio **REFERENCES** sobre esa columna, se descartará la restricción de clave externa de *Reservas*; sin embargo, *no* se descarta la tabla *Reservas*.

Para comprender el motivo de que la norma de SQL decidiera introducir el privilegio **REFERENCES** en vez de permitir simplemente que se empleara el privilegio **SELECT** en esta situación, considérese lo que ocurre si la definición de *Reservas* especifica la opción **NO ACTION** con la clave externa —puede que se impida que José, el dueño de *Barcos*, elimine una fila de *Barcos* debido a que una fila de *Reservas* hace referencia a esa fila de *Barcos*—. Dar a Francisco, el creador de *Reservas* el derecho a restringir de esa manera las actualizaciones de

Barcos va mucho más allá de permitirle simplemente leer los valores de Barcos, que es todo lo que permite el privilegio SELECT.

14.4 CONTROL OBLIGATORIO DE ACCESO

Los mecanismos de control discrecional de acceso, aunque por lo general efectivos, tienen algunos puntos débiles. En concreto, son vulnerables a los esquemas de tipo *caballo de Troya*, en los que usuarios no autorizados con malas intenciones pueden engañar a usuarios autorizados para que desvelen datos delicados. Por ejemplo, supóngase que el estudiante Diego Taimado desea tener acceso a las tablas de notas del profesor Justino Confiado. Diego hace lo siguiente:

- Crea una tabla nueva denominada MíoTodoMío y concede a Justino (quien, por supuesto, está felizmente ignorante de toda esta atención) el privilegio INSERT sobre esa tabla.
- Modifica el código de alguna aplicación del SGBD que Justino utilice a menudo para que haga un par de cosas más: en primer lugar, leer la tabla Notas y, a continuación, escribir el resultado en MíoTodoMío.

Luego descansa, espera a que se copien las notas en MíoTodoMío y luego deshace las modificaciones de la aplicación para asegurarse de que Justino no averigua posteriormente de ninguna manera que lo han engañado. Por tanto, pese a que el SGBD aplica todos los controles discretionarios de acceso —sólo se permitió que el código autorizado de Justino tuviera acceso a Notas— se han desvelado datos delicados a un intruso. El hecho de que Diego pudiera modificar de manera furtiva el código de Justino queda fuera del ámbito del mecanismo de control de acceso del SGBD.

Los mecanismos de control obligatorio de acceso están pensados para abordar esos puntos débiles del control discrecional de acceso. El modelo más popular de control obligatorio de acceso, denominado modelo de Bell-LaPadula, se describe en términos de **objetos** (es decir, tablas, vistas, filas, columnas), **sujetos** (es decir, usuarios, programas), **clases de seguridad** y **autorizaciones**. Se asigna a cada objeto de la base de datos una *clase de seguridad*, y a cada sujeto se le concede *autorización* para una clase de seguridad; se denota la clase del objeto o sujeto *A* como *clase(A)*. Las clases de seguridad de cada sistema se organizan de acuerdo con un orden parcial, con una **clase más segura** y una **clase menos segura**. En aras de la sencillez se supondrá que hay cuatro clases: *máximo secreto* (*MS*), *secreto* (*S*), *confidencial* (*C*) y *no clasificado* (*N*). En este sistema *MS > S > C > N*, donde *A > B* significa que los datos de la clase *A* son más delicados que los de la clase *B*.

El modelo de Bell-LaPadula impone dos restricciones a todas las operaciones de lectura y de escritura en los objetos de la base de datos:

1. **Propiedad de seguridad sencilla.** Sólo se permite que el sujeto *S* lea el objeto *O* si $\text{clase}(S) \geq \text{clase}(O)$. Por ejemplo, los usuarios autorización *MS* pueden leer tablas con autorización *C*, pero no se permite que los usuarios con autorización *C* lean tablas con clasificación *MS*.
2. **Propiedad *.** Sólo se permite que el sujeto *S* escriba el objeto *O* si $\text{clase}(S) \leq \text{clase}(O)$. Por ejemplo, los usuarios con autorización *S* sólo pueden escribir objetos con clasificación *S* o *MS*.

Si también se especifican controles discrecionales de acceso, estas reglas representan restricciones adicionales. Por tanto, para leer o escribir objetos de la base de datos los usuarios deben tener los privilegios necesarios (obtenidos mediante órdenes GRANT) *y* las clases de seguridad de usuario y objeto deben satisfacer estas restricciones. Consideremos el modo en que un mecanismo de control así podría haber derrotado a Diego Taimado. Se podría clasificar la tabla Notas como *S*, se podría conceder a Justino autorización para *S* y se le podría conceder a Diego Taimado una autorización de menor nivel (*C*). Diego sólo puede crear objetos de clasificación *C* o inferior; por tanto, la tabla MíoTodoMío puede tener, como máximo, la clasificación *C*. Cuando el programa de aplicación que se ejecuta en nombre de Justino (y, por tanto, con autorización *S*) intenta copiar Notas en MíoTodoMío, no se le permite hacerlo, ya que *clase(MíoTodoMío) < clase(aplicación)* y se viola la propiedad *.

14.4.1 Relaciones multinivel y poliinstanciación

Para aplicar políticas de control obligatorio de acceso en SGBD relacionales hay que asignar una clase de seguridad a cada objeto de la base de datos. Los objetos pueden hallarse en la granularidad de las tablas, las filas o, incluso, valores concretos de las columnas. Supongamos que se asigna una clase de seguridad a cada fila. Esta situación conduce al concepto de **tabla multinivel**, que son tablas con la sorprendente propiedad de que usuarios con autorizaciones de seguridad diferentes ven conjuntos distintos de filas cuando tienen acceso a la misma tabla.

Considérese el caso de la tabla Barcos de la Figura 14.3. Tanto los usuarios con autorización *S* como los que tienen autorización *MS* obtienen filas en la respuesta cuando piden ver todas las filas de Barcos. Los usuarios con autorización *C* sólo obtiene la segunda fila, mientras que los usuarios con autorización *N* no obtienen ninguna.

<i>idb</i>	<i>nombreb</i>	<i>color</i>	Clase de seguridad
101	Salsa	Rojo	<i>S</i>
102	Pinto	Marrón	<i>C</i>

Figura 14.3 La instancia *B1* de Barcos

Se ha definido que la tabla Barcos tiene *idb* como clave principal. Supóngase que un usuario con autorización *C* desea introducir la fila $\langle 101, \text{Picante}, \text{Escarlata}, \text{C} \rangle$. Se plantea un dilema:

- Si se permite la inserción, dos filas diferentes de la tabla tendrán la clave 101.
- Si no se permite la inserción porque se viola la restricción de clave principal, el usuario que intenta insertar la fila nueva, que tiene autorización *C*, puede deducir que hay un barco con *idb=101* cuya clase de seguridad es superior a . Esta situación pone en peligro el principio de que los usuarios no deben poder deducir ninguna información sobre objetos que tengan una clasificación de seguridad más elevada.

Este dilema se resuelve tratando realmente la clasificación de seguridad como parte de la clave. Así, se permite que la inserción siga adelante y se modifica la instancia de la tabla como puede verse en la Figura 14.4.

<i>idb</i>	<i>nombreb</i>	<i>color</i>	Clase de seguridad
101	Salsa	Rojo	<i>S</i>
101	Picante	Escarlata	<i>C</i>
102	Pinto	Marrón	<i>C</i>

Figura 14.4 La instancia *B1* tras la inserción

Los usuarios con autorizaciones *C* o *N* sólo ven las filas de Picante y Pinto, pero los que tienen autorizaciones *S* o *MS* ven todas las filas. Las dos filas con *idb=101* se pueden interpretar de dos maneras: sólo existe realmente la fila de clasificación más elevada (Salsa, con clasificación *S*) o existen las dos y su presencia se revela a los usuarios en función de su nivel de autorización. La elección de la interpretación adecuada depende de los desarrolladores y de los usuarios de la aplicación.

La presencia de objetos de datos que parecen tener valores diferentes para usuarios con distintas autorizaciones (por ejemplo, el barco con *idb 101*) se denomina **poliinstanciación**. Si se consideran las clasificaciones de seguridad asociadas con cada columna, la intuición subyacente a la poliinstanciación se puede generalizar de manera sencilla, pero se deben abordar algunos detalles adicionales. Hay que destacar que el principal inconveniente de los esquemas de control obligatorio de acceso es su rigidez; las políticas las definen los administradores de los sistemas y los mecanismos de clasificación no son lo bastante flexibles. Todavía está por lograr una combinación satisfactoria de controles de acceso discrecionales y obligatorios.

14.4.2 Canales ocultos, niveles de seguridad del DoD

Aunque el SGBD haga cumplir el esquema de control obligatorio de acceso que se acaba de analizar, la información puede pasar de los niveles de clasificación más elevados a los inferiores por medios indirectos, denominados **canales ocultos**. Por ejemplo, si una transacción tiene acceso a datos de más de un lugar de un SGBD distribuido, hay que coordinar las acciones en todos esos sitios. Puede que el proceso en un sitio tenga una autorización más baja (por ejemplo, *C*) que el de otro (digamos, *S*), y todos los procesos deben acordar comprometerse antes de que se pueda comprometer la transacción. Este requisito se puede aprovechar para pasar información con clasificación *S* al proceso con autorización *C*: se invoca repetidamente la transacción y el proceso con autorización *C* siempre acuerda comprometerse, mientras que el proceso con autorización *S* acuerda comprometerse si desea transmitir un bit 1 y no acuerda hacerlo si desea transmitir un bit 0.

De esta manera (realmente tortuosa) la información con autorización *S* se puede enviar a procesos con autorización *C* en forma de corriente de bits. Este canal oculto es una violación indirecta de lo que se pretende con la propiedad *. Se pueden encontrar fácilmente ejemplos adicionales de canales ocultos en las bases de datos estadísticas, que se analizan en el Apartado 14.6.2.

Los fabricantes de SGBD han comenzado recientemente a implementar mecanismos de control obligatorio de acceso (aunque no forman parte de la norma de SQL) debido a que el Departamento de Defensa (Department of Defense, DoD) de los Estados Unidos de América

Sistemas actuales. Hay disponibles SGBDR que soportan los controles discrecionales en el nivel *C2* y los obligatorios en el *B1*. DB2 de IBM, Informix, SQL Server de Microsoft, Oracle 8 y ASE de Sybase soportan características de SQL para el control discrecional de acceso. En general, no soportan el control obligatorio; Oracle ofrece una versión de su producto con soporte para el control obligatorio de acceso.

exige que sus sistemas lo soporten. Los requisitos del DoD se pueden describir en términos de los **niveles de seguridad** *A*, *B*, *C* y *D*, de los cuales el *A* es el más seguro y el *D* el menos seguro.

El nivel *C* exige el soporte del control discrecional de acceso. Se divide en los subniveles *C1* y *C2*; *C2* también exige cierto grado de responsabilidad mediante procedimientos como la comprobación de los inicios de sesión y las trazas de auditoría. El nivel *B* exige el soporte del control obligatorio de acceso. Se subdivide en los niveles *B1*, *B2* y *B3*. El nivel *B2* exige también la identificación y la eliminación de los canales ocultos. El nivel *B3* también exige el mantenimiento de las trazas de auditoría y el nombramiento de un **administrador de seguridad** (generalmente, pero no es obligatorio, el DBA). El nivel *A*, el más seguro, exige una prueba matemática de que el mecanismo de seguridad hace que se cumpla la política de seguridad.

14.5 SEGURIDAD PARA LAS APLICACIONES DE INTERNET

Cuando se tiene acceso a un SGBD desde una ubicación segura, se puede confiar en un mecanismo sencillo de contraseñas para autenticar a los usuarios. Sin embargo, supongamos que nuestro amigo Samuel desea formular el pedido de un libro por Internet. Esto plantea algunos desafíos peculiares: Samuel no es ni siquiera un usuario conocido (a menos que sea un cliente que repite). Desde el punto de vista de Amazon, se tiene a una persona que pide un libro y ofrece pagar con una tarjeta de crédito registrada a nombre de Samuel pero, ¿esa persona es realmente Samuel? Desde el punto de vista de Samuel, ve un formulario que pide la información de la tarjeta de crédito pero, ¿se trata realmente de una parte auténtica del sitio Web de Amazon y no una aplicación canalla diseñada para engañarlo y que revele el número de su tarjeta de crédito?

Este ejemplo ilustra la necesidad de un enfoque de la autenticación más sofisticado que los mecanismos sencillos de contraseñas. Las técnicas de cifrado ofrecen la base de la autenticación moderna.

14.5.1 Cifrado

La idea básica es aplicar un **algoritmo de cifrado** a los datos, mediante una **clave de cifrado** especificada por el usuario o por el DBA. El resultado del algoritmo es la versión cifrada de los datos. También hay un **algoritmo de descifrado**, que toma los datos cifrados y una **clave de descifrado** como datos y devuelve los datos originales. Sin la clave de descifrado

DES y AES. La norma DES, adoptada en 1977, tiene una clave de cifrado de 56 bits. Con el tiempo, los ordenadores han llegado a ser tan rápidos que, en 1999, se empleó un procesador construido al efecto y una red de PC para violar la DES en menos de un día. El sistema probaba 245 mil millones de claves por segundo cuando se halló la correcta. Se estima que se puede construir un dispositivo hardware para que rompa la DES en menos de cuatro horas por menos de un millón de euros. Pese a la creciente preocupación sobre su vulnerabilidad, DES se sigue utilizando mucho. En 2000 se adoptó como nueva norma de cifrado (simétrico) una sucesora de la DES, denominada **norma de cifrado avanzada (Advanced Encryption Standard, AES)**. AES tiene tres tamaños posibles de la clave: 128, 192 y 256 bits. Con un tamaño de clave de 128 bits hay más de $3 \cdot 10^{38}$ claves AES posibles, lo que es del orden de 10^{24} veces más que el número de claves DES de 56 bits. Supóngase que se pudiera crear un ordenador tan rápido como para romper la DES en 1 segundo. Ese ordenador tendría que calcular durante unos 149 billones de años para poder romper una clave AES de 128 bits. (Los expertos creen que el Universo tiene menos de 20000 millones de años de antigüedad.)

correcta, el algoritmo de descifrado produce resultados sin sentido. Se da por supuesto que los propios algoritmo de cifrado y de descifrado son de conocimiento público pero, al menos, una (dependiendo del esquema de cifrado) de las claves es secreta.

En el **cifrado simétrico** también se utiliza la clave de cifrado como clave de descifrado. La **norma de cifrado de datos** (Data Encryption Standard, DES) de ANSI, que se utiliza desde 1977, es un ejemplo bien conocido de cifrado simétrico. Emplea un algoritmo de cifrado que consiste en sustituciones y permutaciones de caracteres. El principal punto débil del cifrado simétrico es que hay que comunicar la clave a todos los usuarios autorizados, lo que aumenta la probabilidad de que llegue a ser conocida por intrusos (por ejemplo, por un mero error humano).

Otro enfoque del cifrado, denominado **cifrado de clave pública**, se ha hecho cada vez más popular en los últimos años. El esquema de cifrado propuesto por Rivest, Shamir y Adleman, denominado RSA, es un ejemplo bien conocido de cifrado de clave pública. Cada usuario autorizado tiene una **clave de cifrado pública**, conocida por todo el mundo, y una **clave de descifrado** privada, que sólo conoce ese usuario. Dado que las claves de descifrado privadas sólo son conocidas por sus propietarios, se evita el punto débil de la DES.

Un aspecto fundamental del cifrado de clave pública es el modo en que se eligen las claves de cifrado y de descifrado. Técnicamente, los algoritmos de cifrado de clave pública se basan en la existencias de **funciones de un solo sentido**, cuyas inversas son computacionalmente muy difíciles de determinar. El algoritmo RSA, por ejemplo, se basa en la observación de que, aunque es sencillo comprobar si un número dado es primo, determinar los factores primos de un número no primo resulta extremadamente difícil. (Determinar los factores primos de un número de más de cien cifras puede tardar años de tiempo de CPU en los ordenadores más rápidos disponibles hoy en día.)

Ahora se esbozará la idea subyacente al algoritmo RSA, suponiendo que los datos que se van a cifrar son el número entero E . Para escoger una clave de cifrado y otra de descifrado para un usuario dado se elige primero un número entero muy grande, G , más grande que el

Razón del funcionamiento de RSA. El punto principal del esquema es que resulta sencillo calcular d dados c , p y q , pero *muy* complicado calcular d si sólo se dan e y L . A su vez, esta dificultad depende del hecho de que resulta difícil determinar los factores primos de L , que resulta que son p y q . *Una advertencia:* es una creencia extendida que la factorización es difícil, pero no hay ninguna prueba de que ésta lo sea. Tampoco hay prueba alguna de que la factorización sea la única manera de romper RSA; es decir, calcular d a partir de e y de L .

entero más grande que sea necesario cifrar¹. Luego se selecciona un número c como clave de cifrado y se calcula la clave de descifrado d basándose en c y en G ; el modo de hacerlo es fundamental para este enfoque, como se verá en breve. Tanto G como c se hacen públicos y los utiliza el algoritmo de cifrado. Sin embargo, d se mantiene secreta y es necesaria para el descifrado.

- La función de cifrado es $S = E^c \bmod G$.
- La función de descifrado es $E = S^d \bmod G$.

Se ha escogido que G sea el producto de dos números primos diferentes de gran tamaño (por ejemplo, de 1024 bits), $p * q$. La clave de cifrado c es un número elegido al azar entre 1 y L que es primo con respecto a $(p - 1) * (q - 1)$. La clave de descifrado d se calcula de modo que $d * c = 1 \bmod ((p - 1) * (q - 1))$. Dadas estas opciones, se pueden utilizar los resultados de la teoría de números para probar que la función de descifrado recupera el mensaje original a partir de su versión cifrada.

Una propiedad muy importante de los algoritmos de cifrado y de descifrado es que el papel de las claves de cifrado y de descifrado se puede invertir:

$$\text{descifrar}(\text{d}, (\text{cifrar}(\text{c}, \text{G}))) = \text{G} = \text{descifrar}(\text{c}, (\text{cifrar}(\text{d}, \text{G})))$$

Dado que muchos protocolos se basan en esta propiedad, de aquí en adelante sólo nos referiremos a claves públicas y privadas (dado que ambas claves pueden utilizarse para cifrado y para descifrado).

Al introducir el cifrado en el contexto de la autenticación se advirtió que resulta una herramienta fundamental para aplicar la seguridad. Los SGBD pueden utilizar el *cifrado* para proteger información en situaciones en las que no resulten adecuados los mecanismos de seguridad normales del SGBD. Por ejemplo, puede que un intruso robe las cintas que contienen ciertos datos o pinche una línea de comunicaciones. Al almacenar y transmitir los datos en forma cifrada, el SGBD garantiza que esos datos robados no resulten comprensibles para el intruso.

14.5.2 Servidores de certificación: el protocolo SSL

Supóngase que se asocia una clave pública y una clave de descifrado con Amazon. Cualquiera, por ejemplo, el usuario Samuel, puede enviar a Amazon un pedido si lo cifra empleando la

¹Los mensajes que se van a cifrar se descomponen en bloques tales que cada uno de ellos pueda tratarse como un entero menor que G .

clave pública de Amazon. Sólo Amazon puede descifrar ese pedido secreto, ya que el algoritmo exige la clave privada de Amazon, que sólo conoce la propia Amazon.

Esto depende de la capacidad de Samuel de averiguar de manera fiable la clave pública de Amazon. Varias compañías actúan de **autoridades de certificación**, como, por ejemplo, Verisign. Amazon genera la clave de cifrado pública c_A (y la clave de cifrado privada correspondiente) y la envía a Verisign. Verisign emite entonces un **certificado** para Amazon que contiene la información siguiente:

(Verisign, Amazon, <https://www.amazon.com>, c_A)

El certificado se cifra empleando la propia clave *privada* de Verisign, que es conocida por (es decir, se almacena en) Internet Explorer, Netscape Navigator y otros navegadores.

Cuando Samuel llega al sitio Web de Amazon y desea formular un pedido, su navegador, que ejecuta el protocolo SSL², pide al servidor el certificado de Verisign. El navegador valida entonces el certificado descifrándolo (empleando la clave pública de Verisign) y comprobando que el resultado es un certificado con el nombre Verisign, y que el URL que contiene es el del servidor con el que está comunicándose. (Obsérvese que los intentos de falsificar certificados fallarán, ya que los certificados se cifran empleando la clave privada de Verisign, que sólo conoce Verisign.) A continuación el navegador genera una **clave de sesión** aleatoria, la cifra empleando la clave pública de Amazon (que ha obtenido del certificado validado y en la que, por lo tanto, confía) y la envía al servidor de Amazon.

A partir de este momento el servidor de Amazon y el navegador pueden emplear la clave de sesión (que los dos conocen y confían en que sólo ellos la conocen) y un protocolo de cifrado *simétrico* como AES o DES para intercambiar de manera segura mensajes cifrados: Los mensajes los cifra el remitente y los descifra el receptor empleando la misma clave de sesión. Los mensajes cifrados viajan por Internet y pueden ser interceptados, pero no se pueden descifrar sin la clave de sesión. Resulta útil considerar el motivo de necesitar una clave de sesión; después de todo, el navegador podría haberse limitado a cifrar la solicitud original de Samuel empleando la clave pública de Amazon y a enviarla de manera segura al servidor de Amazon. El motivo es que, sin la clave de sesión, el servidor de Amazon no tiene manera de devolver la información al navegador de manera segura. Otra ventaja de las claves de sesión es que el cifrado simétrico es mucho más rápido en términos de cálculo que el cifrado de clave pública. La clave de sesión se descarta al término de la sesión.

Por tanto, Samuel puede estar tranquilo de que sólo Amazon puede ver la información que escribe en el formulario que le muestra el servidor de Amazon y la información que le devuelve en las respuestas del servidor. Sin embargo, en este momento, Amazon no tiene ninguna garantía de que el usuario que ejecuta el navegador sea realmente Samuel y no alguien que haya robado su tarjeta de crédito. Generalmente, los comerciantes aceptan esta situación, que también surge cuando los clientes formulan pedidos por teléfono.

Si se desea estar seguro de la identidad de los usuarios, se puede lograr exigiendo además que inicien una sesión. En este ejemplo, Samuel debe abrir en primer lugar una cuenta con Amazon y seleccionar una contraseña. (La identidad de Samuel se comprueba en primer lugar llamándolo por teléfono para comprobar la información de la cuenta o enviándole un mensaje a una dirección de correo electrónico; en este último caso, todo lo que se comprueba

²Los navegadores emplean el protocolo SSL si el URL de destino comienza por *https*.

es que el propietario de la cuenta es la persona con la dirección de correo electrónico dada.) Siempre que visite el sitio Web y Amazon necesite comprobar su identidad, lo redirigirá a un formulario de inicio de sesión *tras emplear SSL para definir una clave de sesión*. La contraseña que se escriba se transmitirá de manera segura cifrándola con la clave de sesión.

Otro inconveniente más de este enfoque es que ahora Amazon conoce el número de la tarjeta de crédito de Samuel, y él debe confiar en que Amazon no haga mal uso de ese número. El protocolo de **transacciones electrónicas seguras** aborda esta limitación. Ahora todos los clientes deben obtener un certificado con sus propias claves públicas y privadas, y cada transacción implica al servidor de Amazon, al navegador del cliente y al servidor de una tercera parte de confianza, como Visa para las transacciones con tarjeta de crédito. La idea básica es que el navegador cifre la información que no atañe a la tarjeta de crédito mediante la clave pública de Amazon y la relativa a la tarjeta de crédito mediante la clave pública de Visa y las envíe al servidor de Amazon, que transfiere la información sobre la tarjeta de crédito (que no puede descifrar) al servidor de Visa. Si el servidor de Visa aprueba esa información, la transacción sigue adelante.

14.5.3 Firmas digitales

Supóngase que Emilio, que trabaja para Amazon, y Blanca, que trabaja para McGraw-Hill, necesitan mantener comunicación en relación con el inventario. Se puede utilizar el cifrado de clave pública para crear **firmas digitales** para los mensajes. Es decir, se pueden cifrar los mensajes de modo que, si Emilio recibe un mensaje supuestamente remitido por Blanca, puede comprobar que es cierto (además de poder descifrarlo) y, además, *probar* que es de Blanca de McGraw-Hill, aunque el mensaje se envíe desde una cuenta de Hotmail mientras Blanca está de viaje. De manera parecida, Blanca puede autenticar al originador de los mensajes de Emilio.

Si Emilio cifra los mensajes para Blanca empleando la clave pública de ella, y viceversa, pueden intercambiar información de manera segura pero no pueden autenticar al remitente. Quien quisiera suplantar a Blanca podría utilizar su clave pública para enviar un mensaje a Emilio, fingiendo ser Blanca.

No obstante, un empleo inteligente del esquema de cifrado permite que Emilio compruebe si el mensaje lo envió realmente Blanca. Blanca cifra el mensaje empleando su clave *privada* y luego cifra el resultado utilizando la clave pública de Emilio. Cuando Emilio recibe ese mensaje, primero lo descifra empleando su clave privada y luego descifra el resultado empleando la clave pública de Blanca. Este paso genera el mensaje original sin cifrar. Además, Emilio puede estar seguro de que el mensaje lo redactó y lo cifró Blanca, ya que un falsificador no podría conocer su clave privada y, sin ella, el resultado final no tendría sentido, en vez de ser un mensaje legible. Además, como ni siquiera Emilio conoce la clave privada de Blanca, Blanca no puede afirmar que Emilio haya falsificado el mensaje.

Si la autenticación del usuario es el objetivo y ocultar el mensaje no es importante, se puede reducir el coste del cifrado empleando una **firma de mensaje**. La firma se obtiene aplicando al mensaje una función de un solo sentido (por ejemplo, un esquema de asociación) y es considerablemente más pequeña. La firma se cifra como en el enfoque básico de la firma digital y se envía junto con el mensaje completo y sin cifrar. El receptor puede comprobar

el remitente de la firma como se acaba de describir y validar el propio mensaje aplicando la función de un solo sentido y comparando el resultado con la firma.

14.6 OTROS ASPECTOS DE LA SEGURIDAD

La seguridad es un campo muy amplio y su tratamiento en este libro es necesariamente limitado. Este apartado aborda brevemente otros aspectos importantes de la seguridad.

14.6.1 Papel de los administradores de bases de datos

Los administradores de bases de datos (Database Administrators, DBA) desempeñan un importante papel en la aplicación de los aspectos relativos a la seguridad del diseño de las bases de datos. Junto con los propietarios de los datos, el DBA contribuye también al desarrollo de la política de seguridad. El DBA tiene una cuenta especial, que se llama **cuenta del sistema**, y es responsable de la seguridad global del sistema. En concreto, el DBA trata con lo siguiente:

1. **Creación de nuevas cuentas.** Se debe asignar un identificador de autorización y una contraseña a cada usuario o grupo de usuarios nuevo. Obsérvese que los programas de aplicación que tienen acceso a la base de datos tienen el mismo identificador de autorización que el usuario que ejecuta el programa.
2. **Aspectos del control obligatorio.** Si el SGBD soporta el control obligatorio —algunos sistemas personalizados para aplicaciones con requisitos de seguridad muy estrictos (por ejemplo, los datos militares) ofrecen ese soporte— el DBA debe asignar clases de seguridad a todos los objetos de la base de datos y autorizaciones de seguridad a cada identificador de autorización de acuerdo con la política de seguridad escogida.

El DBA también es responsable del mantenimiento de la **traza de auditoría** que, básicamente, es el registro de las actualizaciones con el identificador de autorización (del usuario que ejecutó la transacción) añadido a cada entrada del registro. Este registro no es más que una extensión menor del mecanismo de registro empleado para la recuperación en caso de fallo. Además, puede que el DBA decida mantener un registro de *todas* las acciones, incluidas las operaciones de lectura, llevadas a cabo por los usuarios. El análisis de esos históricos del modo en que se ha tenido acceso al SGBD puede ayudar a evitar violaciones de la seguridad mediante la identificación de patrones extraños antes de que un intruso acabe teniendo éxito en sus intentos de acceso, o puede ayudar a localizar al intruso una vez detectada la violación de la seguridad.

14.6.2 Seguridad en las bases de datos estadísticas

Las **bases de datos estadísticas** contienen información concreta de personas o acontecimientos, pero están pensadas para permitir sólo consultas estadísticas. Por ejemplo, si se tuviera una base de datos estadística de información sobre marineros, se permitirían consultas estadísticas sobre la categoría media, la edad máxima, etcétera, pero no sobre marineros concretos. La seguridad de estas bases de datos plantea nuevos problemas, ya que es posible

deducir información protegida (como la categoría de un marinero concreto) a partir de las respuestas a consultas estadísticas permitidas. Esas posibilidades de deducción representan canales ocultos que pueden poner en peligro la política de seguridad de la base de datos.

Supóngase que el marinero Pedro Furtivo desea conocer la categoría del Almirante Trompa, el apreciado presidente del club náutico, y resulta que sabe que Trompa es el marinero más viejo del club. Pedro formula repetidamente consultas de la forma “¿Cuántos marineros hay cuya edad es mayor que X ? ” para varios valores de X , hasta que la respuesta es 1. Evidentemente, ese marinero es Trompa, el marinero más viejo. Obsérvese que cada una de esas consultas es una consulta estadística válida y está permitida. Sea el valor de X en este momento, digamos 65. Ahora Pedro formula la consulta, “¿Cuál es la categoría máxima de los marineros cuya edad es mayor que 65? ” Una vez más, esta consulta está permitida, ya que se trata de una consulta estadística. Sin embargo, la respuesta a la consulta revela a Pedro la categoría de Trompa, y se viola la política de seguridad de la base de datos.

Una manera de evitar estas violaciones es exigir que cada consulta deba implicar, al menos, a un número mínimo, digamos N , de filas. Con una elección razonable de N Pedro no podrá aislar la información sobre Trompa, ya que la consulta sobre la categoría máxima fallaría. Esta restricción, no obstante, es fácil de obviar. Formulando repetidamente consultas de la forma “¿Cuántos marineros hay cuya edad sea mayor que X ? ”, hasta que el sistema rechace una de ellas, Pedro identifica un conjunto de N marineros que incluye a Trompa. Sea el valor de X en este momento 55. Ahora, Pedro puede formular dos consultas:

- “¿Cuál es la suma de las categorías de todos los marineros cuya edad es mayor que 55? ” Dado que N marineros tienen más de 55 años, esta consulta está permitida.
- “¿Cuál es la suma de las categorías de todos los marineros, distintos de Trompa, cuya edad es mayor que 55 y la del marinero Pedro? ” Dado que el conjunto de los marineros cuyas categorías se suman incluye ahora a Pedro en lugar de a Trompa, pero es igual en todo lo demás, el número de marineros implicado sigue siendo N , y la consulta también está permitida.

De las respuestas a estas dos consultas, digamos A_1 y A_2 , Pedro, que conoce su propia categoría, puede calcular fácilmente la de Trompa como $A_1 - A_2 + \text{categoría de Pedro}$.

Pedro tuvo éxito porque pudo formular dos consultas que implicaban en gran parte a los mismos marineros. El número de filas examinadas en común por dos consultas se denomina su **intersección**. Si se impusiera un límite al valor de la intersección permitida entre dos consultas cualesquiera formuladas por el mismo usuario, se podrían frustrar las intenciones de Pedro. En realidad, un usuario verdaderamente malvado (y paciente) suele poder averiguar la información sobre personas concretas aunque el sistema imponga a las consultas un número mínimo de filas implicadas (N) y una intersección máxima permitida (M), pero el número de consultas necesarias para hacerlo crece en proporción a N/M . Se puede intentar limitar también el número total de consultas que puede formular cada usuario, pero dos usuarios seguirían pudiendo conspirar para violar las medidas de seguridad. Si se mantiene un registro de toda la actividad (incluidos los accesos sólo de lectura), se pueden detectar esos patrones de consultas, teóricamente antes de que se produzca ninguna violación de la seguridad. Este análisis debería dejar claro, sin embargo, que es difícil hacer que se cumplan las medidas de seguridad en las bases de datos estadísticas.

14.7 ESTUDIO DE UN CASO DE DISEÑO: LA TIENDA EN INTERNET

Volvamos al estudio de nuestro caso y a nuestros amigos de TiposBD para considerar los aspectos de seguridad. Hay tres grupos de usuarios: clientes, empleados y el propietario de la librería. (Por supuesto, también está el administrador de la base de datos, que tiene acceso universal a todos los datos y es responsable del funcionamiento normal del sistema de bases de datos.)

El propietario de la librería tiene privilegios completos sobre todas las tablas. Los clientes pueden consultar la tabla Libros y formular pedidos en línea, pero no deben tener acceso a los registros ni a los pedidos de otros usuarios. TiposBd restringe el acceso de dos maneras. En primer lugar, diseña una página Web sencilla con varios formularios parecida a la página que puede verse en la Figura 7.1 del Capítulo 7. Esto permite a los clientes remitir pequeños conjuntos de solicitudes válidas sin darles la posibilidad de tener acceso directo al SGBD subyacente mediante una interfaz de SQL. En segundo lugar, TiposBD emplea las características de seguridad del SGBD para limitar el acceso a los datos delicados.

La página Web permite que los clientes consulten la relación Libros por número de ISBN, por nombre del autor y por título del libro. La página Web también tiene dos botones. El primero recupera una lista de todos los pedidos de cada cliente que todavía no se han atendido completamente. El segundo botón muestra una lista de todos los pedidos completados para ese cliente. Obsérvese que los clientes no pueden especificar consultas de SQL auténticas a través de Web, sino tan sólo escribir algunos parámetros en un formulario para obtener una consulta de SQL generada de manera automática. Todas las consultas generadas mediante los datos introducidos en el formulario tienen una cláusula WHERE que incluye el atributo *idc* del cliente correspondiente, y la evaluación de las consultas generadas mediante los dos botones exige el conocimiento del número de identificación del cliente. Dado que todos los clientes tienen que iniciar sesión en el sitio Web antes de navegar por el catálogo, la lógica comercial (que se examina en el Apartado 7.7) debe conservar la información de estado sobre cada cliente (es decir, el número de identificación del cliente) durante toda la visita de ese cliente al sitio Web.

El segundo paso es configurar la base de datos para limitar el acceso de acuerdo a las necesidades de conocimiento de cada grupo de usuarios. TiposBD crea una cuenta **cliente** especial que tiene los privilegios siguientes:

```
SELECT ON Libros, PedidosNuevos, PedidosViejos, ListasNuevosPedidos,
       ListasViejosPedidos
INSERT ON PedidosNuevos, PedidosViejos, ListasNuevosPedidos, ListasViejosPedidos
```

Los empleados deben poder añadir nuevos libros al catálogo, actualizar las existencias de los libros, revisar los pedidos de los clientes si es necesario y actualizar toda la información relativa a los clientes *excepto la relativa a las tarjetas de crédito*. De hecho, los empleados ni siquiera deben poder ver el número de las tarjetas de crédito de los clientes. Por tanto, TiposBD crea la vista siguiente:

```
CREATE VIEW InfoCliente (idc,nombrec,dirección)
AS SELECT C.idc, C.nombrec, C.dirección
```

```
FROM Clientes C
```

TiposBd concede a la cuenta `empleado` los privilegios siguientes:

```
SELECT ON InfoCliente, Libros,
       PedidosNuevos, PedidosViejos, ListasNuevosPedidos, ListasViejosPedidos
INSERT ON InfoCliente, Libros,
       PedidosNuevos, PedidosViejos, ListasNuevosPedidos, ListasViejosPedidos
UPDATE ON InfoCliente, Libros,
       PedidosNuevos, PedidosViejos, ListasNuevosPedidos, ListasViejosPedidos
DELETE ON Libros, PedidosNuevos, PedidosViejos, ListasNuevosPedidos,
       ListasViejosPedidos
```

Obsérvese que los empleados pueden modificar `InfoCliente` e, incluso, insertar tuplas en ella. Esto es posible porque tienen los privilegios necesarios y, además, la vista es actualizable y se le puede insertar información. Aunque parezca razonable que los empleados puedan actualizar la dirección de los clientes, parece raro que puedan insertar tuplas en `InfoCliente` aunque no puedan ver información relacionada acerca de los clientes (por ejemplo, el número de la tarjeta de crédito) en la tabla `Clientes`. El motivo de esto es que la tienda desea poder aceptar por teléfono pedidos de clientes primerizos sin pedirles por ese medio la información sobre la tarjeta de crédito. Los empleados pueden insertar información en `InfoCliente`, lo que crea un nuevo registro de `Clientes` sin la información sobre la tarjeta de crédito, y los clientes pueden facilitar posteriormente el número de la tarjeta de crédito mediante una interfaz Web. (Evidentemente, el pedido no se envía hasta que lo hayan hecho.)

Además, hay problemas de seguridad cuando un usuario inicia sesión en el sitio Web por primera vez empleando el número de identificación de cliente. El envío del número sin cifrar por Internet es un riesgo para la seguridad y se debe emplear un protocolo seguro como SSL.

Empresas como CyberCash o DigiCash ofrecen soluciones de pago para comercio electrónico que incluyen incluso el *dinero en efectivo electrónico*. El estudio del modo de incorporar estas técnicas en el sitio Web cae fuera del ámbito de este libro.

14.8 PREGUNTAS DE REPASO

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

- ¿Cuáles son los objetivos principales del diseño de aplicaciones de bases de datos seguras? Explíquense los términos *secreto*, *integridad*, *disponibilidad* y *autenticación*. (**Apartado 14.1**)
- Explíquense los términos *política de seguridad* y *mecanismo de seguridad* y la manera en que están relacionados. (**Apartado 14.1**)
- ¿Cuál es la principal idea subyacente al *control discrecional de acceso*? ¿Cuál es la idea subyacente al *control obligatorio de acceso*? ¿Cuáles son las ventajas comparativas de estos dos enfoques? (**Apartado 14.2**)
- Describanse los privilegios reconocidos en SQL. En especial: `SELECT`, `INSERT`, `UPDATE`, `DELETE` y `REFERENCES`. Para cada privilegio, indíquese quién lo adquiere de manera automática para cada tabla. (**Apartado 14.3**)

- ¿Cómo se identifica a los propietarios de privilegios? En especial, analíicense los *identificadores* y los *roles de autorización*. (**Apartado 14.3**)
- ¿Qué es un *grafo de autorizaciones*? Explíquense las órdenes de SQL GRANT y REVOKE en términos de sus consecuencias sobre dicho grafo. En especial, analícese lo que ocurre cuando los usuarios transmiten los privilegios que reciben de otros. (**Apartado 14.3**)
- Examíñese la diferencia entre tener un privilegio sobre una tabla y tenerlo sobre una vista definida sobre esa tabla. En especial, la manera en que un usuario puede tener un privilegio (digamos, SELECT) sobre una vista sin tenerlo también sobre todas las tablas subyacentes. ¿Quién debe tener los privilegios correspondientes sobre todas las tablas subyacentes a la vista? (**Apartado 14.3.1**)
- ¿Qué son los *objetos*, los *sujetos*, las *clases de seguridad* y las *autorizaciones* en el control obligatorio de acceso? Examínense las restricciones de Bell-LaPadula en términos de estos conceptos. Concretamente, defínanse la *propiedad sencilla de seguridad* y la *propiedad **. (**Apartado 14.4**)
- ¿Qué es un ataque de *caballo de Troya* y cómo puede poner en peligro el control discrecional de acceso? Explíquese el modo en que el control obligatorio de acceso protege contra los ataques de caballos de Troya. (**Apartado 14.4**)
- ¿Qué significan los términos *tabla multinivel* y *poliinstanciación*? Explíquese su relación y el modo en que surgen en el contexto del control obligatorio de acceso. (**Apartado 14.4.1**)
- ¿Qué son los *canales ocultos* y cómo pueden surgir cuando se aplican tanto el control discrecional de acceso como el obligatorio? (**Apartado 14.4.2**)
- Analíicense los niveles de seguridad del DoD para los sistemas de bases de datos. (**Apartado 14.4.2**)
- Explíquese el motivo de que un mero mecanismo de contraseñas sea insuficiente para la autenticación de los usuarios que tienen acceso remoto a una base de datos, por ejemplo, por Internet. (**Apartado 14.5**)
- ¿Cuál es la diferencia entre *cifrado simétrico* y *clave pública*? Dense ejemplos de algoritmos de cifrado bien conocidos de ambos tipos. ¿Cuál es la principal debilidad del cifrado simétrico y cómo se aborda en el cifrado de clave pública? (**Apartado 14.5.1**)
- Analícese la elección de claves de cifrado y de descifrado en el cifrado de clave pública y el modo en que se emplean para cifrar y descifrar datos. Explíquese el papel de las *funciones de un solo sentido*. ¿Qué seguridad se tiene de que el esquema RSA no se pueda poner en peligro? (**Apartado 14.5.1**)
- ¿Qué son las *autoridades de certificación* y por qué son necesarias? Explíquese el modo en que se emiten y se validan los *certificados* para los sitios mediante navegadores que emplean el *protocolo SSL*; analícese el papel de la *clave de sesión*. (**Apartado 14.5.2**)

- Si un usuario conecta con un sitio que emplea el protocolo SSL, explíquese el motivo de que siga teniendo que iniciar una sesión. Explíquese el empleo de SSL para proteger las contraseñas y otra información delicada que se intercambia. ¿Qué es el *protocolo para transacciones electrónicas seguras*? ¿Cuál es su valor añadido respecto de SSL? (**Apartado 14.5.2**)
- Las *firmas digitales* facilitan el intercambio seguro de mensajes. Explíquese lo que son y el modo en que superan el mero cifrado de mensajes. Examínese el empleo de las *firmas de mensaje* para reducir el coste del cifrado. (**Apartado 14.5.3**)
- ¿Cuál es el papel de los administradores de bases de datos con respecto a la seguridad? (**Apartado 14.6.1**)
- Examínense el resto de agujeros de seguridad introducidos en las *bases de datos estadísticas*. (**Apartado 14.6.2**)

EJERCICIOS

Ejercicio 14.1 Responda brevemente a las preguntas siguientes:

1. Explíquese la intuición subyacente a las dos reglas del modelo de Bell-LaPadula.
2. Dese un ejemplo del modo en que se pueden emplear los canales ocultos para derrotar al modelo de Bell-LaPadula.
3. Dese un ejemplo de poliinstanciación.
4. Describase una situación en la que los controles de acceso obligatorios eviten una violación de la seguridad que no se pueda prevenir mediante controles discrecionales.
5. Describase una situación en la que sean necesarios los controles de acceso discretionales para hacer que se cumpla una política de seguridad que no se pueda aplicar empleando sólo controles obligatorios.
6. Si un SGBD soporta ya los controles de acceso discretionales y los obligatorios, ¿hace falta el cifrado?
7. Explíquese la necesidad de cada uno de los límites siguientes en los sistemas de bases de datos estadísticas:
 - (a) Un máximo para el número de consultas que puede plantear cada usuario.
 - (b) Un mínimo para el número de tuplas implicadas en la respuesta a cada consulta.
 - (c) Un máximo para la intersección de dos consultas (es decir, para el número de tuplas que examinan ambas consultas).
8. Explíquese el empleo de las pistas de auditoría, con especial referencia a los sistemas de bases de datos estadísticas.
9. ¿Cuál es el papel de los DBA con respecto a la seguridad?
10. Describase el AES y su relación con el DES.
11. ¿Qué es el cifrado de clave pública? ¿En qué se diferencia del enfoque de cifrado adoptado en la Norma de Cifrado de Datos (Data Encryption Standard, DES) y en qué aspectos es mejor que la DES?
12. Explíquese el modo en que las compañías que ofrecen servicios en Internet pueden emplear técnicas basadas en el cifrado para proteger sus procesos de admisión de pedidos. Analícese el papel de DES, AES, SSL, SET y las firmas digitales. Búsquese en la web para averiguar más sobre las técnicas relacionadas, como el *dinero electrónico*.

Ejercicio 14.2 Usted es el DBA de la Compañía de Juguetes Fabulosos y crea una relación denominada Empleados con los campos *nombree*, *dep* y *sueldo*. Por motivos relacionados con las autorizaciones también se definen las vistas NombresEmpleados (con *nombree* como único atributo) e InfoDep con los campos *dep* y *sueldomedio*. Esta última muestra el sueldo medio de cada departamento.

1. Muéstrense las instrucciones de definición de vistas para NombresEmpleados y para InfoDep.
2. ¿Qué privilegios se deben conceder a los usuarios que sólo necesiten conocer el sueldo medio de cada departamento para los departamentos Juguetes y AC?
3. Desea autorizar a su secretaria a despedir gente (probablemente le diga a quién debe despedir, pero desea poder delegar esta tarea), a comprobar si alguien figura como empleado y a comprobar el sueldo medio de cada departamento. ¿Qué privilegios le debe conceder?
4. Continuando con la situación anterior, no desea que su secretaria pueda conocer el sueldo de cada empleado. ¿Garantiza esto su respuesta a la pregunta anterior? Concrete: ¿Cabe la posibilidad de que su secretaria descubra el sueldo de *algunos* empleados (dependiendo del conjunto real de tuplas) o puede descubrir siempre el sueldo de cualquier individuo cuando lo deseé?
5. Desea conceder a su secretaria la autoridad para permitir a otras personas la lectura de la vista NombresEmpleados. Muéstrese la orden correspondiente.
6. Su secretaria define dos nuevas vistas mediante la vista NombresEmpleados. La primera se denomina NombresAaR y se limita a seleccionar los nombres que comienzan con una letra comprendida entre la A y la R. La segunda se denomina CuantosNombres y cuenta el número de nombres. Está tan contento con este logro que decide conceder a su secretaria el derecho a insertar tuplas en la vista NombresEmpleados. Muestre la orden correspondiente y describa los privilegios que tiene su secretaria una vez ejecutada esa orden.
7. Su secretaria permite a Teodoro leer la relación NombresEmpleados y luego abandona la empresa. Posteriormente usted revoca los privilegios de su secretaria. ¿Qué ocurre con los privilegios de Teodoro?
8. Dese un ejemplo de actualización de vista para el esquema anterior que no se pueda llevar a cabo mediante actualizaciones de Empleados.
9. Usted decide tomarse unas largas vacaciones y, para asegurarse de que se puede afrontar las posibles emergencias, desea autorizar a su jefe José a que lea y modifique las relaciones Empleados y NombresEmpleados (por supuesto, José también debe poder delegar su autoridad, ya que se halla demasiado alto en la jerarquía de gestión como para hacer ningún trabajo real). Muéstrense las instrucciones de SQL correspondientes. ¿Puede leer José la vista InfoDep?
10. Tras volver de sus (maravillosas) vacaciones, usted ve una nota de José que indica que ha autorizado a su secretario Miguel a leer la relación Empleados. Usted desea revocar el privilegio **SELECT** de Miguel sobre Empleados, pero no los derechos que le concedió a José, ni siquiera temporalmente. ¿Puede hacer esto en SQL?
11. Más adelante se da cuenta de que José ha estado bastante ocupado. Ha definido una vista denominada TodosNombres mediante la vista NombresEmpleados y una relación denominada NombresPersonal a la que él tiene acceso (pero usted no) y concedido a su secretario Miguel el derecho a leer la vista TodosNombres. Miguel ha transmitido ese derecho a su amiga Susana. Usted decide que, aun a costa de molestar a José por la retirada de algunos de sus privilegios, no tiene más remedio que retirar a Miguel y a Susana el derecho a ver sus datos. ¿Qué instrucción **REVOKE** ejecutaría? ¿Qué derechos tiene José sobre Empleados una vez ejecutada esa instrucción? ¿Qué vistas se descartan a consecuencia de ello?

Ejercicio 14.3 Usted es pintor y tiene una tienda en Internet en la que vende sus pinturas directamente al público. Le gustaría que los clientes pagaran sus compras con tarjeta de crédito y desea garantizar que esas transacciones electrónicas estén protegidas.

Supóngase que María desea adquirir su reciente pintura de la Peña del Arcipreste de Hita. Responda a las preguntas siguientes.

1. ¿Cómo puede garantizar que el usuario que está comprando la pintura es verdaderamente María?

2. Explíquese el modo en que SSL garantiza que la comunicación del número de la tarjeta de crédito está protegida. ¿Cuál es el papel de las autoridades de certificación en este caso?
3. Suponga que desea que María pueda comprobar que todos sus mensajes de correo electrónico proceden realmente de usted. ¿Cómo puede autenticar sus mensajes sin cifrar el texto real?
4. Suponga que sus clientes también pueden negociar el precio de determinadas pinturas y que María desea negociar el precio de la pintura del Monasterio del Escorial. Le gustaría que el texto de esta comunicación fuera confidencial entre María y usted. Explíquense las ventajas e inconvenientes de los diferentes métodos de cifrado de su comunicación con María.

Ejercicio 14.4 Considérense los Ejercicios 6.6 a 6.9 del Capítulo 6. Para cada ejercicio, identifíquense los datos que serían accesibles para los diferentes grupos de usuarios y escríbanse las instrucciones de SQL para hacer que se cumpla esa política de control de accesos.

Ejercicio 14.5 Considérense los Ejercicios 7.7 a 7.9 del Capítulo 7. Para cada ejercicio, examíñese dónde pueden resultar apropiados el cifrado, el protocolo SSL y las firmas digitales.



EJERCICIOS BASADOS EN PROYECTOS

Ejercicio 14.6 ¿Hay algún soporte para las vistas o las autorizaciones en Minibase?

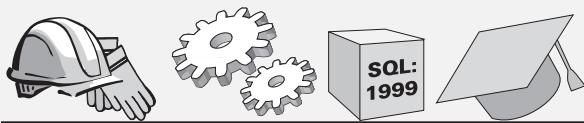
NOTAS BIBLIOGRÁFICAS

El mecanismo de autorización del Sistema R, que tuvo gran influencia en el paradigma GRANT y REVOKE de SQL, se describe en [222]. [145] ofrece un buen tratamiento general de la criptografía y se puede encontrar una introducción a la seguridad de las bases de datos en [86] y en [292]. La seguridad de las bases de datos estadísticas se investiga en varios trabajos, incluidos [144] y [112]. La seguridad de varios niveles se analiza en varios trabajos, como [262, 308, 418, 431].

Una referencia clásica en criptografía es el libro de Schneier [396]. Diffie y Hellman propusieron la primera técnica criptográfica de clave pública [151]. El ampliamente utilizado esquema de cifrado RSA lo introdujeron Rivest, Shamir y Adleman [379]. AES está basado en el algoritmo Rijndael de Daemen y Rijmen [132]. Hay muchos libros de introducción al protocolo SSL, como [375] y [444]. Se puede hallar más información sobre las firmas digitales en el libro de Ford y Baum [184].

PARTE V

TEMAS ADICIONALES



15

SISTEMAS DE BASES DE DATOS DE OBJETOS

- ¿Qué son los sistemas de bases de datos de objetos y qué características nuevas soportan?
- ¿A qué tipos de aplicaciones favorecen?
- ¿Qué clases de tipos de datos pueden definir los usuarios?
- ¿Qué son los tipos abstractos de datos y cuáles son sus ventajas?
- ¿Qué es la herencia de tipos y por qué resulta útil?
- ¿Cuáles son las consecuencias de introducir identificadores de objetos en las bases de datos?
- ¿Cómo se pueden utilizar las características nuevas en el diseño de bases de datos?
- ¿Cuáles son los nuevos retos para la implementación?
- ¿Qué diferencia los SGBD relacionales orientados a objetos y los SGBD orientados a objetos?
- **Conceptos fundamentales:** tipos de datos definidos por los usuarios, tipos estructurados, tipos colección; abstracción de datos, métodos, encapsulación; herencia, vinculación precoz y tardía de los métodos, jerarquías de colecciones; identidad de los objetos, tipos referencia, igualdad superficial y profunda.

en colaboración con Joseph M. Hellerstein
Universidad de California-Berkeley

Usted conoce mis métodos, Watson. Aplíquelos.

— Arthur Conan Doyle, *Memorias de Sherlock Holmes*

Los sistemas relacionales de bases de datos soportan un pequeño conjunto fijo de tipos de datos (por ejemplo, enteros, fechas, cadenas de caracteres) que se han demostrado adecuados

para los dominios de aplicación tradicionales, como el procesamiento de datos administrativos. En muchos dominios de aplicación, sin embargo, se deben manejar tipos de datos mucho más complejos. Generalmente esos datos complejos se han guardado en sistemas de archivos del SO o en estructuras de datos especializadas, en vez de en el SGBD. Entre los ejemplos de dominios con datos complejos están el diseño y el modelado asistido por ordenador (computer-aided design and modeling, CAD/CAM) los repositorios multimedia y la gestión de documentos.

A medida que crece la cantidad de datos, las muchas características ofrecidas por los SGBD se vuelven más atractivas y, finalmente, necesarias —por ejemplo, el tiempo reducido para el desarrollo de aplicaciones, el control de la concurrencia y la recuperación de datos, el soporte de los índices y las posibilidades de consulta—. Para dar soporte a esas aplicaciones, los SGBD deben admitir tipos de datos complejos. Los conceptos orientados a objetos han influido poderosamente en los esfuerzos por mejorar el soporte de las bases de datos a los datos complejos y han llevado al desarrollo de sistemas de bases de datos orientados a objetos, que se estudian en este capítulo.

Los sistemas de bases de datos orientados a objetos se han desarrollado siguiendo dos caminos diferentes:

- **Sistemas de bases de datos orientados a objetos.** Los sistemas de bases de datos orientados a objetos se propusieron como alternativa a los sistemas relacionales y están dirigidos a dominios de aplicación en los que los objetos complejos desempeñan un papel central. Este enfoque está muy influido por los lenguajes de programación orientados a objetos y se pueden considerar un intento de añadir funcionalidad de los SGBD a los lenguajes de programación. El Grupo de Gestión de Bases de Datos de Objetos (Object Database Management Group, ODMG) ha desarrollado un **modelo de datos orientado a objetos (Object Data Model, ODM)** y un **lenguaje de consulta orientado a objetos (Object Query Language, OQL)** normalizados, que son el equivalente de la norma SQL para los sistemas relacionales de bases de datos.
- **Sistemas de bases de datos relacionales orientados a objetos.** Los sistemas de bases de datos relacionales orientados a objetos se pueden considerar un intento de ampliar los sistemas relacionales de bases de datos con la funcionalidad necesaria para soportar una clase más amplia de aplicaciones y, en muchos sentidos, ofrecer un puente entre los paradigmas relacional y orientado a objetos. La norma SQL:1999 amplía SQL para que incorpore el soporte al modelo de datos relacional orientado a objetos.

En este capítulo se emplearán acrónimos para los sistemas de gestión de bases de datos relacionales, orientados a objetos y relacionales orientados a objetos (respectivamente, **SGBDR**, **SGBDOO** y **SGBDROO** —en inglés, RDBMS, OODBMS y ORDBMS—). Los SGBDROO serán la parte principal de este capítulo y se destacará el modo en que se pueden contemplar como un desarrollo de los SGBDR en lugar de un paradigma completamente diferente, como ejemplifica la evolución de SQL:1999.

Nos concentraremos en el desarrollo de los conceptos fundamentales en lugar de presentar SQL:1999; algunas de las características que se estudiarán no están incluidas en SQL:1999. Pese a todo, se han decidido subrayar los conceptos relevantes para SQL:1999 y sus probables extensiones futuras. También se ha intentado ser consistente con SQL:1999 en cuanto a la notación, aunque ocasionalmente pueda diferenciarse ligeramente en aras de la claridad. Es

importante reconocer que los principales conceptos analizados son comunes a los SGBDROO y a los SGBDOO; se estudiará el modo en que están soportados en la norma ODL/OQL propuesta para los SGBDOO en el Apartado 15.9.

Los fabricantes de SGBDR, como IBM, Informix y Oracle, están añadiendo funcionalidad SGBDROO (en diferentes grados) a sus productos, y es importante reconocer la manera en que el corpus de conocimiento existente acerca del diseño y la implementación de bases de datos relacionales se pueden aprovechar para trabajar con las ampliaciones SGBDROO. También es importante comprender los retos y las oportunidades que suponen estas ampliaciones para los usuarios, diseñadores e implementadores de bases de datos.

En este capítulo los Apartados del 15.1 al 15.6 presentan los conceptos orientados a objetos. Los conceptos estudiados en estos apartados son comunes para los SGBDOO y los SGBDROO. Se comenzará con la presentación de un ejemplo en el Apartado 15.1 que ilustra el motivo de que las ampliaciones del modelo relacional sean necesarias para abordar algunos de los nuevos dominios de aplicación. Este ejemplo se utilizará a lo largo del capítulo. Se analizará el empleo de constructores de tipos para el soporte de los tipos de datos estructurados definidos por los usuarios en el Apartado 15.2. Las operaciones soportadas por estos nuevos tipos de datos se considerarán en el Apartado 15.3. A continuación se examinarán la encapsulación de datos y los tipos abstractos de datos en el Apartado 15.4. La herencia y otros problemas relacionados, como el enlace de métodos y las jerarquías de colecciones se tratan en el Apartado 15.5. Luego se considerarán los objetos y su identidad en el Apartado 15.6.

Se estudiará el modo de sacar ventaja de los nuevos conceptos orientados a objetos para llevar a cabo el diseño de bases de datos SGBDROO en el Apartado 15.7. En el Apartado 15.8 se estudian algunos de los nuevos desafíos de implementación planteados por los sistemas relacionales orientados a objetos. Se analizarán ODL y OQL, las normas para los SGBDOO, en el Apartado 15.9, y luego se presentará una breve comparación entre los SGBDROO y los SGBDOO en el Apartado 15.10.

15.1 EJEMPLO MOTIVADOR

Como ejemplo concreto de la necesidad de los sistemas relacionales orientados a objetos nos centraremos en un nuevo problema de procesamiento de datos empresariales que es, a la vez, más difícil y (a nuestro entender) más entretenido que la contabilidad de euros y céntimos de décadas pasadas. Hoy en día las empresas de sectores como el entretenimiento están en el negocio de la venta de *bits*; sus activos corporativos básicos no son productos tangibles sino, más bien, artefactos de software como el vídeo y el sonido.

Imaginemos una empresa ficticia, la Compañía del Pequeño Entretenimiento, un gran grupo empresarial de Hollywood cuyo principal activo es un conjunto de personajes de dibujos animados, especialmente el delicioso e internacionalmente amado Gusano Heriberto. Pequeño tiene varias películas del Gusano Heriberto, muchas de las cuales se exhiben continuamente en cines de todo el mundo. Pequeño también consigue mucho dinero con las licencias de la imagen, de la voz y de fragmentos de vídeo de Heriberto para diferentes finalidades: muñecos, videojuegos, patrocinios de productos, etcétera. La base de datos de Pequeño se utiliza para gestionar los registros de ventas y alquileres de los diferentes productos relacionados con

La norma SQL/MM. SQL/MM es una norma emergente que parte de los nuevos tipos de datos de SQL:1999 para definir ampliaciones de SQL:1999 que facilitan el manejo de tipos de datos multimedia complejos. SQL/MM es una norma con varias partes. La Parte 1, el Marco de SQL/MM identifica los conceptos de SQL:1999 que constituyen la base de las ampliaciones de SQL/MM. Cada una de las otras partes aborda un tipo concreto de datos complejos: **Full Text** (texto completo), **Spatial** (espacial), **Still Image** (imagen fija) y **Data Mining** (minería de datos). SQL/MM anticipa que estos tipos complejos nuevos se podrán emplear en las columnas de las tablas como valores de los campos.

Heriberto, así como los datos de vídeo y de sonido que componen las muchas películas de Heriberto.

15.1.1 Nuevos tipos de datos

El problema fundamental al que se enfrentan los diseñadores de la base de datos de Pequeño es que tienen que dar soporte a tipos de datos considerablemente más variados que los disponibles en los SGBD relacionales:

- **Tipos de datos definidos por los usuarios.** Entre los activos de Pequeño están la imagen, la voz y los fragmentos de vídeo de Heriberto, que deben guardarse en la base de datos. Para manejar estos tipos nuevos hay que poder representar estructuras más variadas. (Véase el Apartado 15.2.) Además, se necesitan funciones especiales para la manipulación de esos objetos. Por ejemplo, puede que se desee escribir funciones que generen una versión comprimida de una imagen, o una imagen de menor resolución. Al ocultar los detalles de la estructura de datos mediante las funciones que capturan el comportamiento, se consigue la *abstracción de los datos*, lo que lleva a un diseño del código más limpio. (Véase el Apartado 15.4.)
- **Herencia.** A medida que crece el número de tipos de datos es importante aprovechar lo que hay en común entre ellos. Por ejemplo, tanto las imágenes comprimidas como las de baja resolución son, en algún nivel, simplemente imágenes. Por tanto, es deseable que se *hereden* algunas características de los objetos de imagen a la hora de definir (y luego manipular) objetos de imágenes comprimidas y objetos de imágenes de baja resolución. (Véase el Apartado 15.5.)
- **Identidad de los objetos.** Dado que algunos de los nuevos tipos de datos contienen ejemplares de tamaño muy grande (por ejemplo, vídeos), es importante no guardar copias de los objetos; en su lugar, se deben guardar *referencias*, o *punteros*, a esos objetos. A su vez, esto subraya la necesidad de dar a los objetos una *identidad de objeto* única, que se puede utilizar para hacer referencia a ellos, o “apuntarlos”, desde cualquier otra parte de los datos. (Véase el Apartado 15.6.)

¿Cómo se pueden abordar estos problemas en los SGBDR? En los sistemas relacionales actuales se pueden guardar imágenes, vídeos, etcétera como BLOB. Los **objetos binarios de gran tamaño (binary large object, BLOB)** no son más que largas cadenas de bytes,

Objetos de gran tamaño. SQL:1999 incluye un tipo de datos nuevo denominado **LARGE OBJECT** (objeto de gran tamaño) o **LOB**, con dos variantes denominadas **BLOB** (objeto grande binario) y **CLOB** (objeto de gran tamaño de caracteres). Esto normaliza el soporte de los objetos grandes existente en muchos SGBD relacionales actuales. Los LOB no se pueden incluir en las claves principales, **GROUP BY** ni en las cláusulas **ORDER BY**. Se pueden comparar empleando la igualdad, las desigualdades y las operaciones de subcadenas de caracteres. Cada LOB tiene un **localizador** que, básicamente, es un identificador único y permite que se manipulen los LOB sin demasiadas copias.

Los LOB se suelen guardar aparte de los registros de datos en cuyos campos aparecen. DB2 de IBM, Informix, SQL Server de Microsoft, Oracle 8 y ASE de Sybase soportan LOB.

y el soporte por parte del SGBD consiste en guardar y recuperar BLOB de modo que los usuarios no se tengan que preocupar por el tamaño del BLOB; cada BLOB puede ocupar varias páginas, a diferencia de los atributos tradicionales. El resto del procesamiento de los BLOB lo tiene que llevar a cabo el programa de aplicación del usuario, en el lenguaje anfitrión en el que se haya incorporado el código de SQL. Esta solución no resulta eficiente, ya que resulta necesario recuperar todos los BLOB de cada conjunto, aunque la mayor parte de los mismos pudiera excluirse de la respuesta mediante la aplicación de funciones definidas por el usuario (dentro del SGBD). Tampoco resulta satisfactorio desde el punto de vista de la consistencia de los datos, ya que la semántica de los datos depende ahora mucho del código de aplicación del lenguaje anfitrión y el SGBD no puede hacer que se cumpla.

En cuanto a los tipos estructurados y la herencia, sencillamente no tienen soporte en el modelo relacional. Resulta obligado asignar los datos con esas estructuras complejas a conjuntos de tablas planas. (Se vieron ejemplos de estas asignaciones cuando se estudió la traducción de los diagramas ER con herencia a relaciones en el Capítulo 2.)

Esta aplicación exige claramente características no disponibles en el modelo relacional. Como ilustración de las mismas, la Figura 15.1 presenta las instrucciones del LDD de SQL:1999 para parte del esquema SGBDROO de Pequeño, usado en los ejemplos posteriores. Aunque el LDD sea muy parecido al de los sistemas relacionales tradicionales, algunas diferencias importantes destacan las nuevas posibilidades de modelado de datos de los SGBDROO. Un vistazo rápido a las instrucciones del LDD bastará por ahora; se estudiarán con detalle en el apartado siguiente, tras la presentación de algunos de los conceptos fundamentales que nuestra aplicación de ejemplo sugiere que son necesarios en los SGBD de la próxima generación.

15.1.2 Manipulación de los nuevos datos

Hasta ahora se han descrito los nuevos tipos de datos que se deben guardar en la base de datos Pequeño. Todavía no se ha dicho nada sobre el modo de *emplearlos* en consultas, por lo conviene pasar a estudiar dos consultas que la base de datos de Pequeño debe soportar. La sintaxis de estas consultas no es crítica; basta con comprender lo que expresan. Se volverá más adelante a los detalles de estas consultas.

```

1. CREATE TABLE Fotogramas
   (númfotograma integer, imagen jpeg-image, categoría integer);
2. CREATE TABLE Categorías
   (idc integer, nombre text, precio_contratación float, comentarios text);
3. CREATE TYPE t_cine AS
   ROW(númc integer, nombre text, dirección text, teléfono text)
   REF IS SYSTEM GENERATED;
4. CREATE TABLE Cines OF cine_c REF is idc SYSTEM GENERATED;
5. CREATE TABLE Exhiben
   (película integer, cine REF(cine_c) SCOPE Cines, estreno date,
    retirada date);
6. CREATE TABLE Películas
   (númpelícula integer, título text, protagonistas VARCHAR(25) ARRAY [10],
    director text, presupuesto float);
7. CREATE TABLE Países
   (nombre text, limita polygon, población integer, idioma text);

```

Figura 15.1 Instrucciones LDD de SQL:1999 para el esquema Pequeño

El primer reto viene de la empresa de cereales para desayuno Zueco. Zueco produce un cereal llamado Delirioso y desea contratar una imagen del Gusano Heriberto frente a un sol naciente para incorporarla al diseño de la caja de Delirioso. Se puede expresar la consulta para presentar un conjunto de imágenes posibles y el precio de su contratación en una sintaxis parecida a la de SQL como en la Figura 15.2. Pequeño tiene varios métodos escritos en un lenguaje imperativo como Java y registrados en el sistema de bases de datos. Esos métodos se pueden utilizar en las consultas igual que se utilizan los métodos predefinidos, como $=$, $+$, $-$, $<$, $>$, en los lenguajes relacionales como SQL. El método *miniatura* de la cláusula **Select** genera una versión de tamaño reducido de su imagen de entrada de tamaño completo. El método *es_amanecer* es una función booleana que analiza las imágenes y devuelve el valor *verdadero* si la imagen contiene un sol naciente; el método *es_heriberto* devuelve el valor *verdadero* si la imagen contiene un retrato de Heriberto. La consulta genera el número del código de fotograma, la imagen miniatura y el precio de todos los fotogramas que contienen a Heriberto y a un sol naciente.

```

SELECT F.Nufotograma, miniatura(F.imagen), C.precio_contratación
FROM   Fotogramas F, Categorías C
WHERE  F.categoría = C.idc AND es_amanecer(F.imagen) AND es_heriberto(F.imagen)

```

Figura 15.2 SQL ampliado para la búsqueda de imágenes de Heriberto al amanecer

El segundo reto procede de los ejecutivos de Pequeño. Saben que Delirioso es enormemente popular en el pequeño país de Andorra, por lo que desean asegurarse de que se exhiban varias películas de Heriberto en cines cercanos a Andorra cuando el cereal llegue a las tiendas. Para comprobar la situación actual, los ejecutivos desean averiguar el nombre de todos los cines

que exhiben películas de Heriberto a una distancia máxima de 100 kilómetros de Andorra. La Figura 15.3 muestra esta consulta en una sintaxis parecida a la del SQL.

```
SELECT E.cine->nombre, E.cine->dirección, P.título
FROM Exhiben E, Películas P, Estados E
WHERE E.película = P.númpelícula AND
solapa(E.limita, radio(E.cine->dirección, 100)) AND
E.nombre = 'Andorra' AND 'El Gusano Heriberto' = P.protagonistas[1]
```

Figura 15.3 SQL ampliado para averiguar las películas de Heriberto que se exhiben cerca de Andorra

El atributo *cine* de la tabla *Exhiben* es una referencia a un objeto de otra tabla, que tiene los atributos *nombre*, *dirección* y *ubicación*. Esta referencia a un objeto permite la notación *E.cine->nombre* y *E.cine->dirección*, cada una de las cuales hace referencia a atributos del objeto de *cine_c* al que se hace referencia en la fila *N* de *Exhiben*. El atributo *protagonistas* de la tabla *películas* es el conjunto de nombres de los protagonistas de cada película. El método *radio* devuelve un círculo centrado en su primer argumento con radio igual a su segundo argumento. El método *solapa* comprueba el solapamiento espacial. *Exhiben* y *Películas* se reúnen mediante la cláusula de equirreunión, mientras que *Exhiben* y *Estados* se reúnen mediante la cláusula de solapamiento espacial. Las selecciones de "Andorra" y de las películas que incluyen al "Gusano Heriberto" completan la consulta.

Estas dos consultas relacionales orientadas a objetos son parecidas a las consultas de SQL-92, pero tienen algunas características no habituales:

- **Métodos definidos por el usuario.** Los tipos abstractos definidos por los usuarios se manipulan mediante sus métodos, por ejemplo, *es_heriberto* (Apartado 15.2).
- **Operadores para los tipos estructurados.** Junto a los tipos estructurados disponibles en el modelo de datos los SGBDROO proporcionan los métodos naturales para ellos. Por ejemplo, el tipo **ARRAY** soporta la operación habitual para arrays del acceso a elementos del array mediante la especificación de su valor de índice; *P.protagonistas[1]* devuelve el primer elemento del array de la columna *protagonistas* de la película *P* (Apartado 15.3).
- **Operadores para los tipos referencia.** Los tipos referencia se *desreferencian* mediante una notación de flechas (*->*) (Apartado 15.6.2).

Para resumir los puntos destacados por el ejemplo motivador, los sistemas relacionales tradicionales ofrecen una flexibilidad limitada en cuanto a los tipos de datos disponibles. Los datos se guardan en tablas y el tipo de los valores de cada campo queda limitado a un solo tipo atómico (por ejemplo, entero o cadena de caracteres) con un pequeño conjunto fijo de estos tipos entre los que elegir. Este sistema limitado de tipos se puede ampliar de tres maneras principales: los tipos abstractos de datos definidos por los usuarios, los tipos estructurados y los tipos referencia. En conjunto nos referiremos a estos tipos nuevos como **tipos complejos**. En el resto de este capítulo se considerará el modo en que se puede ampliar un SGBD para que ofrezca soporte a la definición de tipos complejos nuevos y a la manipulación de los objetos de esos tipos nuevos.

15.2 TIPOS DE DATOS ESTRUCTURADOS

SQL:1999 permite que los usuarios definan nuevos tipos de datos, además de los tipos predefinidos (por ejemplo, enteros). En el Apartado 5.7.2 se analizó la definición de nuevos tipos *distintos*. Los tipos distintos siguen dentro del modelo relacional estándar, ya que sus valores deben ser atómicos.

SQL:1999 ha introducido también dos **constructores de tipos** que permiten definir tipos nuevos con estructura interna. Los tipos definidos mediante constructores de tipos se denominan **tipos estructurados**. Esto conduce más allá del modelo relacional, ya que el valor de estos campos ya no tiene necesariamente que ser atómico:

- **ROW($n_1 t_1, \dots, n_n t_n$)**. Tipo que representa una fila, o tupla, de n campos con los campos n_1, \dots, n_n de los tipos t_1, \dots, t_n , respectivamente.
- **base ARRAY [i]**). Tipo que representa un array de (hasta) i elementos del tipo **base**.

El tipo `cine_c` de la Figura 15.1 ilustra el nuevo tipo de datos **ROW**. En SQL:1999 el tipo **ROW** tiene un papel especial, ya que todas las tablas son conjuntos de filas —cada tabla es un conjunto de filas o un multiconjunto de filas—. Los valores de los demás tipos sólo pueden aparecer como valores de los campos.

El campo *protagonistas* de la tabla Películas ilustra el nuevo tipo **ARRAY**. Se trata de un array de hasta 10 elementos, cada uno de los cuales es del tipo **VARCHAR(25)**. Obsérvese que 10 es el número máximo de elementos del array; en un momento dado el array (a diferencia, por ejemplo, de lo que ocurre en C) puede contener menos elementos. Dado que SQL:1999 no soporta los arrays multidimensionales, puede que *vector* hubiera sido un nombre más acertado para el constructor de arrays.

La potencia de los constructores de tipos procede del hecho de que se pueden componer. El tipo fila siguiente contiene un campo que es un array de diez cadenas de caracteres, como máximo:

```
ROW(númpelícula: integer, protagonistas: VARCHAR(25) ARRAY [10])
```

El tipo fila de SQL:1999 es bastante general; sus campos pueden ser de cualquier tipo de datos de SQL:1999. Por desgracia, el tipo array se halla restringido; los elementos de los arrays no pueden ser a su vez arrays. Por tanto, la definición siguiente es ilegal:

```
(integer ARRAY [5]) ARRAY [10]
```

15.2.1 Tipos colección

SQL:1999 sólo soporta los constructores de tipos **ROW** y **ARRAY**. Otros constructores de tipos habituales son

- **listof(base)**. Un tipo que representa una secuencia de elementos del tipo **base**.
- **setof(base)**. Un tipo que representa un *conjunto* de elementos del tipo **base**. Los conjuntos no pueden contener elementos duplicados.
- **bagof(base)**. Un tipo que representa una *bolsa* o *multiconjunto* de elementos del tipo **base**.

Los tipos de datos estructurados de SQL:1999. Varios sistemas comerciales, como DB2 de IBM, UDS de Informix y Oracle 9i soportan los constructores `ROW` y `ARRAY`. Los constructores de tipos `listof`, `bagof` y `setof` no se hallan incluidos en SQL:1999. No obstante, los sistemas comerciales soportan algunos de estos constructores en diferentes grados. Oracle soporta las relaciones anidadas y los arrays, pero no la composición completa de estos constructores. Informix soporta los constructores `setof`, `bagof` y `listof`, permitiendo su composición. El soporte en esta área varía mucho de unos fabricantes a otros.

Los tipos que utilizan `listof`, `ARRAY`, `bagof` o `setof` como constructor de tipos externo se conocen a veces como **tipos colección** o **tipos de datos masivos**.

La falta de soporte para estos tipos colección está reconocida como una debilidad del soporte de SQL:1999 a los objetos complejos y es bastante posible que parte de estos tipos colección se añadan en revisiones futuras de la norma de SQL¹.

15.3 OPERACIONES CON DATOS ESTRUCTURADOS

El SGBD proporciona métodos predefinidos para los tipos definidos mediante los constructores de tipos. Estos métodos son análogos a las operaciones intrínsecas como la suma y la multiplicación de tipos atómicos como los enteros. En este apartado se presentarán los métodos para diversos constructores de tipos y se ilustrará la manera en que las consultas SQL pueden crear y manipular valores con tipos estructurados.

15.3.1 Operaciones con filas

Dado el elemento i , cuyo tipo es $\text{ROW}(n_1\ t_1, \dots, n_n\ t_n)$, el método de extracción de campos permite el acceso al campo concreto n_k mediante la notación de puntos tradicional $i.n_k$. Si los constructores de filas se anidan en la definición de tipo, se pueden anidar los puntos para tener acceso a los campos de la fila anidada; por ejemplo $i.n_k.m_l$. Si se tiene un conjunto de filas, la notación de puntos da como resultado un conjunto. Por ejemplo, si i es una lista de filas, $i.n_k$ da una lista de elementos del tipo t_n ; si i es un conjunto de filas, $i.n_k$ da un conjunto de elementos del tipo t_n .

Esta notación de puntos anidados se denomina a menudo **expresión de ruta**, ya que describe una ruta por la estructura anidada.

15.3.2 Operaciones con arrays

Los tipos para arrays soportan un método de “índice del array” que permite que los usuarios tengan acceso a los elementos del array en unas circunstancias determinadas. Se suele usar la sintaxis de notación polaca inversa con “corchetes”. Dado que el número de elementos puede

¹Según Jim Melton, el editor de la norma de SQL:1999, ya se consideró la inclusión de estos tipos colección, pero se omitieron debido a que se descubrieron algunos problemas con sus especificaciones demasiado tarde para poder corregirlos a tiempo para SQL:1999.

variar, existe el operador (**CARDINALITY**) que devuelve el número de elementos del array. Este número variable de elementos también justifica que un operador concatene dos arrays. El ejemplo siguiente ilustra estas operaciones sobre arrays de SQL:1999.

```
SELECT P.númpelícula, (P.protagonistas || ['Brando', 'Pacino'])
FROM Películas P
WHERE CARDINALITY(P.protagonistas) < 3 AND P.protagonistas[1]='Redford'
```

Para cada película con Redford como primer protagonista² y menos de tres protagonistas, el resultado de la consulta contiene el array de protagonistas de la película concatenado con el array que contiene los dos elementos “Brando” y “Pacino”. Obsérvese el modo de crear un valor de tipo array (que contiene a Brando y a Pacino) mediante el empleo de corchetes en la cláusula SELECT.

15.3.3 Operaciones con otros tipos colección

Aunque en SQL:1999 sólo se soporten los arrays, se espera que las versiones futuras de SQL soporten otros tipos colección, y aquí se estudiarán las operaciones que resultan adecuadas con esos tipos de datos. Este estudio es ilustrativo y no pretende ser exhaustivo. Por ejemplo, también se podría permitir la aplicación de los operadores de agregación *count*, *sum*, *avg*, *max* y *min* a cualquier objeto de tipo conjunto con el tipo base adecuado (por ejemplo, **INTEGER**). También se podrían soportar los operadores para las conversiones de tipos. Por ejemplo, se podrían proporcionar operadores para convertir objetos de tipo multiconjunto en objetos de tipo conjunto mediante la eliminación de los valores duplicados.

Conjuntos y multiconjuntos

Los objetos de tipo conjunto se pueden comparar mediante los métodos tradicionales para conjuntos \subset , \subseteq , $=$, \supseteq , \supset . Los elementos del tipo **setof(nombretipo)** se puede comparar con elementos del tipo **nombretipo** mediante el método \in , como puede verse en la Figura 15.3, que contiene la comparación “*El Gusano Heriberto*” \in *P.protagonistas*. Se pueden combinar dos objetos de tipo conjunto (que tengan elementos del mismo tipo) para formar un nuevo objeto mediante los operadores \cup , \cap y $-$.

Se puede definir para los multiconjuntos todos los métodos para conjuntos, teniendo en cuenta el número de copias de cada elemento. La operación \cup se limita a sumar el número de copias de cada elemento, \cap cuenta el número mínimo de veces que aparece un elemento dado en los dos multiconjuntos de entrada, mientras que $-$ resta el número de veces que aparece cada elemento en el segundo multiconjunto del número de veces que aparece en el primero. Por ejemplo, empleando la semántica de los multiconjuntos $\cup (\{1,2,2,2\}, \{2,2,3\}) = \{1,2,2,2,2,2,3\}$; $\cap (\{1,2,2,2\}, \{2,2,3\}) = \{2,2\}$; y $- (\{1,2,2,2\}, \{2,2,3\}) = \{1,2\}$.

²Obsérvese que el primer elemento de todo array de SQL tiene el valor de índice 1 (no 0, como ocurre en algunos lenguajes).

Listas

Entre las operaciones tradicionales con listas están *head*, que devuelve el primer elemento; *tail*, que devuelve la lista obtenida mediante la eliminación del primer elemento; *prepend*, que toma un elemento y lo inserta como primer elemento de una lista; y *append*, que añade una lista tras otra.

15.3.4 Consultas a colecciones anidadas

Aquí se presentarán algunos ejemplos para ilustrar la manera en que se pueden consultar las relaciones que contienen colecciones anidadas empleando la sintaxis de SQL. En especial, se han estudiado a fondo las ampliaciones del modelo relacional con conjuntos y multiconjuntos anidados y nos centraremos en esos tipos colección.

En este apartado se considerará una variante de la relación Películas de la Figura 15.1, con el campo *protagonistas* definido como `setof` (`VARCHAR[25]`), en vez de como array. Cada tupla describe una película, identificada de manera única mediante *númpelícula*, y contiene un conjunto (de protagonistas de la película) como valor del campo.

El primer ejemplo ilustra el modo en que se pueden aplicar los operadores de agregación a estos conjuntos anidados. Identifica las películas con más de dos protagonistas contando el número de protagonistas, aplicando el operador `CARDINALITY` una vez por tupla de Películas³.

```
SELECT P.númpelícula
  FROM Películas P
 WHERE CARDINALITY(P.protagonistas) > 2
```

La segunda consulta ilustra una operación denominada **desanidamiento**. Considérese la instancia de Películas mostrada en la Figura 15.4; se han omitido los campos *director* y *presupuesto* (incluidos en el esquema de Películas de la Figura 15.1) por simplificar. Puede verse una versión plana de la misma información en la Figura 15.5; por cada película y protagonista de película tenemos una tupla en Películas_plana.

<i>númpelícula</i>	<i>título</i>	<i>protagonistas</i>
98	Casablanca	{Bogart, Bergman}
54	Las lombrices de tierra son jugosas	{Heriberto, Wanda}

Figura 15.4 Una relación anidada, Películas

La consulta siguiente genera la instancia de Películas_plana a partir de Películas:

```
SELECT P.númpelícula, P.título, I AS protagonista
  FROM Películas P, P.protagonistas AS I
```

La variable *P* está vinculada sucesivamente a tuplas de Películas y, para cada valor de *P*, la variable *I* se vincula sucesivamente al conjunto del campo *protagonistas* de *P*. Por otra parte

³SQL:1999 no soporta los valores de conjuntos ni de multiconjuntos, como ya se ha indicado. Si lo hiciera, resultaría natural permitir que se aplicara el operador `CARDINALITY` a un valor de conjunto para que contara el número de elementos; aquí se ha utilizado el operador con esta idea.

<i>númpelícula</i>	<i>título</i>	<i>protagonista</i>
98	Casablanca	Bogart
98	Casablanca	Bergman
54	Las lombrices de tierra son jugosas	Heriberto
54	Las lombrices de tierra son jugosas	Wanda

Figura 15.5 Una versión plana, Películas_plana

puede que se desee generar la instancia de Películas a partir de Películas_plana. La instancia de Películas se puede generar mediante una forma generalizada de la estructura GROUP BY de SQL, como ilustra la consulta siguiente:

```
SELECT P.númpelícula, P.título, set_gen(P.protagonista)
FROM Películas_plana P
GROUP BY P.númpelícula, P.título
```

Este ejemplo introduce el nuevo operador *set_gen* para utilizarlo con GROUP BY, el cual necesita alguna explicación. La cláusula GROUP BY divide la tabla Película_plana ordenándola según el atributo *númpelícula*; todas las tuplas de una partición dada tienen el mismo *númpelícula* (y, por tanto, el mismo *título*). Considérese el conjunto de valores de la columna *protagonista* de una partición dada. En las consultas de SQL-92 este conjunto se debe resumir mediante la aplicación de un operador de agregación como COUNT. Ahora que se permite que las relaciones contengan conjuntos como valores de campo, sin embargo, podemos devolver el conjunto de valores de *protagonista* como valor de campo en una sola tupla de respuesta; la tupla de respuesta contiene también el *númpelícula* de la partición correspondiente. El operador *set_gen* reúne el conjunto de valores de *protagonista* de una partición y crea un objeto con valor de conjunto. Esta operación se denomina **anidamiento**. Se pueden imaginar funciones generadoras parecidas para la creación de multiconjuntos, listas, etcétera. Sin embargo, esos generadores no están incluidos en SQL:1999.

15.4 ENCAPSULACIÓN Y TAD

Considérese la tabla Fotogramas de la Figura 15.1. Tiene una columna *imagen* del tipo *jpeg_image*, que guarda una imagen comprimida que representa un solo fotograma de una película. El tipo *jpeg_image* no está entre los tipos predefinidos del SGBD y lo definió un usuario para que la aplicación Pequeño guarde datos de imágenes comprimidos mediante la norma JPEG. Como ejemplo adicional, la tabla Estados definida en la línea 7 de la Figura 15.1 tiene una columna *limita* del tipo *polygon*, que contiene representaciones de las formas de las fronteras de los países en un mapamundi.

Permitir que los usuarios definan nuevos tipos de datos arbitrarios es una característica fundamental de los SGBDROO. El SGBD permite que los usuarios guarden y recuperen objetos del tipo *jpeg_image*, igual que los objetos de cualquier otro tipo, como puede ser *integer*. Generalmente los nuevos tipos atómicos de datos necesitan que el usuario que los crea defina operaciones específicas para ese tipo de datos. Por ejemplo, se podrían definir operaciones con los tipos de datos de imagen como **comprimir**, **rotar**, **reducir** y **recortar**.

La combinación de un tipo de datos atómicos con sus métodos asociados se denomina **tipo abstracto de datos**, o **TAD**. El SQL tradicional incluye TAD predefinidos, como los enteros (con sus métodos aritméticos asociados) o las cadenas de caracteres (con la igualdad, la comparación y los métodos LIKE). Los sistemas relacionales orientados a objetos incluyen estos TAD y también permiten que los usuarios definan los suyos propios.

La etiqueta *abstracto* se aplica a estos tipos de datos debido a que el sistema de bases de datos no necesita conocer el modo en que se guardan los datos de los TAD ni la manera en que actúan sus métodos. Tan sólo necesita saber los métodos que están disponibles y sus tipos de datos de entrada y de salida. La ocultación de las interioridades de los TAD se denomina **encapsulación**⁴. Obsérvese que, incluso en un sistema relacional, los tipos atómicos como los enteros tiene métodos asociados que los encapsulan. En el caso de los enteros, los métodos estándar para sus TAD son los habituales operadores aritméticos y comparadores. Para evaluar el operador suma con enteros, el sistema de bases de datos no necesita comprender las leyes de la suma —tan sólo necesita saber la manera de invocar el código del operador suma y el tipo de datos que debe esperar como respuesta—.

En los sistemas relacionales orientados a objetos la simplificación debida a la encapsulación resulta fundamental, ya que oculta cualquier distinción sustantiva entre los tipos de datos y permite que se implemente el SGBDROO sin anticipar los tipos de datos y los métodos que los usuarios puedan desear añadir. Por ejemplo, el sistema puede tratar de manera uniforme la suma de enteros y la superposición de imágenes, con las únicas diferencias significativas de que se invoca código diferente para las dos operaciones y de que se espera que ese código devuelva objetos de tipos diferentes.

15.4.1 Definición de métodos

Para registrar un método nuevo para un tipo de datos definido por el usuario se debe escribir el código para el método e informar al sistema de bases de datos de su existencia. El código a escribir depende de los lenguajes soportados por el SGBD y, posiblemente, del sistema operativo en cuestión. Por ejemplo, puede que los SGBDROO manejen código Java en el sistema operativo Linux. En ese caso, el código del método se debe escribir en Java y compilarse en un archivo de código de bytes de Java guardado en un sistema de archivos de Linux. A continuación se emite una orden de registro de métodos en SQL al SGBDROO para que reconozca el nuevo método:

```
CREATE FUNCTION es_amanecer(jpeg_image) RETURNS boolean
AS EXTERNAL NAME '/a/b/c/pequeño.clase' LANGUAGE 'java';
```

Esta instrucción define los aspectos más destacados del método: el tipo del TAD asociado, el tipo devuelto y la ubicación del código. Una vez registrado el método, el SGBD emplea una máquina virtual de Java para ejecutar el código⁵. La Figura 15.6 presenta varias órdenes para el registro de métodos para la base de datos Pequeño.

⁴ Algunos SGBDROO se refieren realmente a los TAD como **tipos opacos**, ya que están encapsulados y, por tanto, no se pueden ver sus detalles.

⁵ En el caso de código compilado no portable —escrito, por ejemplo, en un lenguaje como C++— el SGBD utiliza la capacidad de vinculación dinámica del sistema operativo para vincular el código del método con el sistema de bases de datos para que se pueda invocar.

Ampliaciones empaquetadas de los SGBDROO. El desarrollo de un conjunto de tipos y métodos definidos por el usuario para una aplicación concreta —por ejemplo, la gestión de imágenes— puede suponer una cantidad significativa de trabajo y de conocimiento de un dominio concreto. En consecuencia, la mayor parte de los fabricantes de SGBDROO se asocian con terceros para vender conjuntos empaquetados previamente de TAD para dominios concretos. Informix llama a estas ampliaciones *DataBlades*, Oracle las denomina *Data Cartridges*, IBM las llama *DB2 Extenders*, etcétera. Estos paquetes incluyen el código del método TAD, secuencias de comandos de LDD para automatizar la carga de los TAD en el sistema y, en algunos casos, métodos de acceso especializados para ese tipo de datos. Las ampliaciones TAD empaquetadas son análogas a las bibliotecas de clase disponibles para los lenguajes de programación orientados a objetos: proporcionan un conjunto de objetos que abordan en conjunto una tarea común.

SQL:1999 tiene una ampliación denominada SQL/MM que consta de varias partes independientes, cada una de las cuales especifica una biblioteca de tipo para una clase de datos concreta. Las partes de SQL/MM para Full-Text (texto completo), Spatial (espacial), Still Image (imagen fija) y Data Mining (minería de datos) ya están disponibles o a punto de publicarse.

```

1. CREATE FUNCTION miniatura(jpeg_image) RETURNS jpeg_image
   AS EXTERNAL NAME '/a/b/c/pequeño.clase' LANGUAGE 'java';
2. CREATE FUNCTION es_amanecer(jpeg_image) RETURNS boolean
   AS EXTERNAL NAME '/a/b/c/pequeño.clase' LANGUAGE 'java';
3. CREATE FUNCTION es_heriberto(jpeg_image) RETURNS boolean
   AS EXTERNAL NAME '/a/b/c/pequeño.clase' LANGUAGE 'java';
4. CREATE FUNCTION radio(polygon, float) RETURNS polygon
   AS EXTERNAL NAME '/a/b/c/pequeño.clase' LANGUAGE 'java';
5. CREATE FUNCTION solapa(polygon, polygon) RETURNS boolean
   AS EXTERNAL NAME '/a/b/c/pequeño.clase' LANGUAGE 'java';

```

Figura 15.6 Órdenes para el registro de métodos para la base de datos Pequeño

Las instrucciones para la definición de tipos de datos atómicos definidos por el usuario en el esquema Pequeño se ilustran en la Figura 15.7.

1. CREATE ABSTRACT DATA TYPE jpeg_image
(*longitudinterna* = VARIABLE, *input* = jpeg_in, *output* = jpeg_out);
2. CREATE ABSTRACT DATA TYPE polygon
(*longitudinterna* = VARIABLE, *input* = poly_in, *output* = poly_out);

Figura 15.7 Órdenes para la declaración de tipos atómicos para la base de datos Pequeño

15.5 HERENCIA

En el Capítulo 2 se consideró el concepto de herencia en el contexto del modelo ER y se examinó el modo en que los diagramas ER con herencia se traducían en tablas. En los sistemas de bases de datos orientados a objetos, a diferencia de los sistemas relacionales, la herencia está soportada directamente y permite que se reutilicen y se refinen muy fácilmente las definiciones de tipos. Puede resultar muy útil al modelar clases de objetos parecidas pero ligeramente diferentes. En los sistemas de bases de datos orientados a objetos se puede utilizar la herencia de dos maneras: para reutilizar y refinar los tipos y para crear jerarquías de colecciones de objetos parecidos pero no idénticos.

15.5.1 Definición de tipos con herencia

En la base de datos Pequeño se modelan los cines con el tipo `t_cine`. Pequeño también desea que su base de datos represente una nueva técnica de marketing en el negocio cinematográfico: el *cine-café*, que sirve pizza y otras comidas mientras se proyectan las películas. Los cines-cafés necesitan que se represente información adicional en la base de datos. En concreto, un cine-café es igual que un cine, pero tiene un atributo adicional que representa el menú del cine. La herencia permite capturar esta “especialización” de manera explícita en el diseño de la base de datos con la siguiente instrucción LDD:

```
CREATE TYPE t_cinecafe UNDER t_cine (menú text);
```

Esta instrucción crea un nuevo tipo, `t_cinecafe`, que tiene los mismos atributos y métodos que `t_cine`, y el atributo adicional *menú* del tipo `text`. Los métodos definidos para `t_cine` se aplican a objetos del tipo `t_cinecafe`, pero no al contrario. Se dice que `t_cinecafe` hereda los atributos y métodos de `t_cine`.

Obsérvese que el mecanismo de la herencia no es simplemente una macro para abreviar las instrucciones `CREATE`. Crea una relación explícita en la base de datos entre el **subtipo** (`t_cinecafe`) y el **supertipo** (`t_cine`): *se considera que los objetos del subtipo también son objetos del supertipo*. Este tratamiento significa que cualquier operación que se pueda aplicar al supertipo (métodos u operadores de consulta, como la proyección o la reunión) se puede aplicar también al subtipo. Esto se expresa generalmente en el principio siguiente:

Principio de sustitución. Dado un supertipo *A* y un subtipo *B*, siempre es posible sustituir un objeto del tipo *B* en una expresión legal escrita para objetos del tipo *A* sin producir errores de tipo.

Este principio permite la reutilización fácil del código, ya que las consultas y los métodos escritos para el supertipo se pueden aplicar al subtipo sin modificaciones.

Obsérvese que la herencia también se puede utilizar para tipos atómicos, además de para los tipos de filas. Dado el supertipo `t_imagen` con los métodos `título()`, `número_de_colores()` y `mostrar()`, se puede definir el subtipo `t_imagen_miniatura` para las imágenes pequeñas que hereda los métodos de `t_imagen`.

15.5.2 Métodos de vinculación

Al definir subtipos resulta útil a veces sustituir un método para el supertipo por una versión nueva que opere de manera diferente con el subtipo. Considérese el tipo `t_imagen` y el subtipo `t_imagen_jpeg` de la base de datos Pequeño. Por desgracia, el método `display()` para las imágenes normales no funciona para las imágenes JPEG, que están comprimidas de manera especial. Por tanto, al crear el tipo `t_imagen_jpeg`, se escribe un método `display()` especial para las imágenes JPEG y se registra en el sistema de bases de datos mediante la orden `CREATE FUNCTION`:

```
CREATE FUNCTION display(imagen_jpeg) RETURNS imagen_jpeg
    AS EXTERNAL NAME '/a/b/c/jpeg.clase' LANGUAGE 'java';
```

Registrar un método nuevo con el mismo nombre que un método viejo se denomina **sobrecargar** el nombre del método.

Debido a la sobrecarga, el sistema debe comprender el método que se pretende utilizar en cada expresión concreta. Por ejemplo, cuando el sistema necesita invocar el método `display()` para un objeto del tipo `t_imagen_jpeg`, utiliza el método especializado `display`. Cuando necesita invocar `display` para un objeto del tipo `t_imagen` que no tiene ningún subtipo, invoca el método `display` estándar. El proceso de decidir el método que se debe invocar se denomina **vincular** el método al objeto. En determinadas situaciones esta vinculación se puede llevar a cabo cuando se analiza una expresión (**vinculación precoz**), pero en otros casos no se puede conocer el tipo más específico del objeto hasta el momento de la ejecución, por lo que el método no se puede vincular hasta entonces (**vinculación tardía**). La vinculación tardía añade flexibilidad, pero puede dificultar que el usuario razonne sobre los métodos que se invocan en una expresión de consulta dada.

15.5.3 Jerarquías de colecciones

La herencia de tipos se inventó para los lenguajes de programación orientados a objetos, y nuestro estudio de la herencia hasta este momento se diferencia poco de los que se pueden encontrar en los libros sobre lenguajes orientados a objetos como C++ o Java.

Sin embargo, como los sistemas de bases de datos proporcionan lenguajes de consultas para conjuntos tabulares de datos, los mecanismos procedentes de los lenguajes de programación se mejoran en las bases de datos de objetos para que traten también con tablas y consultas.

IDO. DB2 de IBM, UDS de Informix y Oracle 9i soportan los tipos REF.

En concreto, en los sistemas relacionales orientados a objetos se puede definir una tabla que contenga objetos de un tipo determinado, como la tabla Cines del esquema Pequeño. Dado un nuevo subtipo, como `t_cinecafé`, nos gustaría crear otra tabla `Cine-cafés` para guardar la información sobre los cine cafés. Pero, al escribir consultas sobre la tabla `Cines`, a veces resulta deseable plantear la misma consulta sobre la tabla `Cine-cafés`; después de todo, si se proyectan al exterior las columnas adicionales, las instancias de la tabla `Cine-cafés` se pueden considerar instancias de la tabla `Cines`.

En vez de exigir que el usuario especifique una consulta diferente para cada una de esas tablas, se puede informar al sistema de que hay que tratar la nueva tabla de ese subtipo como parte de una tabla del supertipo, en relación con las consultas sobre la última tabla. En este ejemplo se puede decir:

```
CREATE TABLE Cine-cafés OF TYPE cinecafé_t UNDER Cines;
```

Esta instrucción indica al sistema que las consultas sobre la tabla `Cines` se deben ejecutar realmente sobre todas las tuplas de las tablas `Cines` y `Cine-cafés`. En esos casos, si la definición del subtipo implica la sobrecarga de los métodos, se emplea la vinculación tardía para garantizar que se llame a los métodos apropiados para cada tupla.

En general, la cláusula `UNDER` se puede utilizar para generar un árbol arbitrario de tablas, denominado **jerarquía de colecciones**. Las consultas sobre una tabla concreta `T` de la jerarquía se ejecutan sobre todas las tuplas de `T` y de sus descendientes. Puede que a veces un usuario desee que la consulta sólo se ejecute sobre `T` y no sobre sus descendientes; para conseguir este efecto se puede emplear sintaxis adicional como, por ejemplo, la palabra clave `ONLY`, en la cláusula `FROM` de la consulta.

15.6 OBJETOS, IDO Y TIPOS REFERENCIA

En los sistemas de bases de datos orientados a objetos se puede dar a los objetos de datos un **identificador de objeto (ido)**, que es un valor que es único en la base de datos a lo largo del tiempo. El SGBD es responsable de la generación de los idos y de garantizar que cada ido identifica de manera única a un objeto durante toda su existencia. En algunos sistemas todas las tuplas guardadas en cualquier tabla son objetos y se les asignan idos de manera automática; en otros sistemas el usuario puede especificar las tablas para las que hay que asignar idos a las tuplas. A menudo también hay medios para la generación de idos para estructuras mayores (por ejemplo, tablas) y menores (por ejemplo, instancias de valores de datos como una copia del entero 5 o una imagen JPEG).

El ido de los objetos se puede emplear para hacer referencia a ellos desde cualquier otra parte de los datos. Cada ido tiene un tipo parecido al tipo de los punteros de los lenguajes de programación.

En SQL:1999 se puede dar un ido a todas las tuplas de una tabla definiendo esa tabla en términos de un tipo estructurado y declarando que está asociado con ella un tipo `REF`, como en la definición de la tabla `Cines` de la línea 4 de la Figura 15.1. Compárese esto con la definición

de la tabla Estados de la línea 7; las tuplas de Estados no tienen idos asociados. (SQL:1999 también asigna “idos” a objetos de gran tamaño: se trata del localizador del objeto.)

Los tipos REF tienen valores que son identificadores únicos o idos. SQL:1999 exige que cada tipo REF esté asociado con una tabla concreta. Por ejemplo, la línea 5 de la Figura 15.1 define la columna *cine* del tipo REF(*t_cine*). La cláusula SCOPE especifica que los elementos de esa columna son referencias a filas de la tabla Cines, que se define en la línea 4.

15.6.1 Conceptos de igualdad

La diferencia entre tipos referencia y tipos estructurados sin referencia plantea otro problema: la definición de igualdad. Por definición, dos objetos que tienen el mismo tipo son **iguales en profundidad** si y sólo si:

1. Los objetos son de tipo atómico y tienen el mismo valor.
2. Los objetos son de tipo referencia y el operador *deep equals* es verdadero para los dos objetos a los que se hace referencia.
3. Los objetos son de tipo estructurado y el operador *deep equals* es verdadero para todos los subcomponentes correspondientes de los dos objetos.

Por definición, dos objetos que tienen el mismo tipo referencia son **iguales superficialmente** si los dos hacen referencia al mismo objeto (es decir, las dos referencias utilizan el mismo ido). La definición de igualdad superficial se puede ampliar a objetos de tipo arbitrario tomando la definición de igualdad profunda y sustituyendo *iguales en profundidad* por *iguales superficialmente* en las partes (2) y (3).

Por ejemplo, considérese los objetos complejos ROW(538, *t89*, 6-3-97, 8-7-97) y ROW(538, *t33*, 6-3-97, 8-7-97), cuyo tipo es el tipo de filas de la tabla Exhiben (línea 5 de la Figura 15.1). Estos dos objetos no son iguales superficialmente porque se diferencian en el valor del segundo atributo. Pese a todo, puede que sean iguales en profundidad, si, por ejemplo, los idos *t89* y *t33* hacen referencia a objetos del tipo *t_cine* que tienen el mismo valor; por ejemplo, tuple(54, ‘Majestic’, ‘Real 115’, ‘2556698’).

Aunque puede que dos objetos iguales en profundidad no lo sean superficialmente, como ilustra el ejemplo, por supuesto que dos objetos iguales superficialmente siempre lo son profundamente. La opción predeterminada de igualdad profunda o superficial para los tipos referencia varía de unos sistemas a otros, aunque se suele dar la sintaxis para especificar cualquiera de las semánticas.

15.6.2 Desreferencia de los tipos referencia

Un elemento del tipo referencia REF(*tipobásico*) no es igual que el elemento *tipobásico* al que apunta. Para tener acceso al elemento al que se hace referencia *tipobásico* se proporciona el método predefinido *deref()* junto con el constructor de tipos REF. Por ejemplo, dada una tupla de la tabla Exhiben, se puede tener acceso al campo *nombre* del objeto al que se hace referencia *t_cine* con la sintaxis *Exhiben.deref(cine).nombre*. Dado que las referencias a los tipos de tuplas son comunes, SQL:1999 utiliza un operador de flecha al estilo Java, que

combina una versión en notación polaca inversa del operador para eliminar referencias con un operador punto tipo tupla. Se puede tener acceso al nombre del cine al que se hace referencia con la sintaxis equivalente Exhiben.cine->nombre, como en la Figura 15.3.

En este momento se han tratado todas las ampliaciones para los tipos básicos empleadas en el esquema Pequeño de la Figura 15.1. Se invita al lector a que revise el esquema y a que examine la estructura y contenido de cada tabla y el modo en que se utilizan las características nuevas en las diferentes consultas de ejemplo.

15.6.3 URL e IDO en SQL:1999

Resulta instructivo observar las diferencias entre los URL de Internet y los idos de los sistemas orientados a objetos. En primer lugar, los idos identifican de manera unívoca un solo objeto a lo largo del tiempo (al menos, hasta que se elimina, momento en que el ido queda indefinido), mientras que el recurso Web al que apunta el URL puede cambiar con el tiempo. En segundo lugar, los idos no son más que identificadores y no llevan consigo ninguna información física sobre los objetos que identifican —esto hace posible modificar la ubicación de almacenamiento de los objetos sin modificar los punteros a ellos—. Por el contrario, los URL incluyen direcciones de red y, a menudo, también los nombres del sistema de archivos, lo que significa que, si el recurso identificado por el URL se tiene que trasladar a otro archivo o dirección de la red, todos los vínculos con ese recurso pasan a ser incorrectos o necesitan un mecanismo de “entrega”. En tercer lugar, los idos los generan de manera automática los SGBD para cada objeto, mientras que los URL los generan los usuarios. Dado que los usuarios generan los URL, a menudo incluyen en ellos información semántica a través del nombre de la máquina, del directorio o del archivo; esto puede volverse confuso si las propiedades del objeto varían con el tiempo.

Para los URL las eliminaciones pueden resultar problemáticas: conducen al famoso error “404 Página no encontrada”. Para los idos SQL:1999 permite escribir REFERENCES ARE CHECKED como parte de la cláusula SCOPE y escoger entre varias acciones a realizar cuando se elimine el objeto al que hace referencia. Se trata de una ampliación directa de la integridad referencial que incluye a los idos.

15.7 DISEÑO DE BASES DE DATOS PARA SGBDROO

La amplia variedad de tipos de datos de los SGBDROO ofrece a los diseñadores de bases de datos muchas oportunidades para llevar a cabo diseños más naturales o eficientes. En este apartado se ilustrarán las diferencias entre el diseño de bases de datos SGBDR y SGBDROO mediante varios ejemplos.

15.7.1 Tipos colección y TAD

El primer ejemplo implica varias sondas espaciales, cada una de las cuales graba un vídeo de manera continua. Se asocia con cada sonda una sola corriente de vídeo y, aunque esa corriente se ha recogido a lo largo de un periodo de tiempo determinado, se supone que ahora se trata de un objeto completo asociado con la sonda. Durante el periodo de tiempo a lo largo del cual

se recogió el vídeo, la ubicación de la sonda se registró de manera periódica (esa información se puede añadir fácilmente a la cabecera de la corriente de vídeo que cumpla la norma MPEG). La información asociada con cada sonda tiene tres partes: (1) un *IDENTIFICADOR de sonda* que identifica de manera única cada sonda (2) una *corriente de vídeo* y (3) una *secuencia de posición* de pares de $\langle \text{hora}, \text{posición} \rangle$. ¿Qué tipo de esquema de base de datos se debe utilizar para guardar esta información?

Diseño de base de datos SGBDR

En los SGBDR hay que guardar cada corriente de vídeo como BLOB y cada secuencia de ubicación como una tupla de una tabla. A continuación puede verse un posible diseño de bases de datos SGBDR:

Sondas(*ids: integer, hora: timestamp, lat: real, long: real,*
cámara: string, vídeo: BLOB)

Hay una sola tabla llamada Sondas y tiene varias filas para cada sonda. Cada una de esas filas tiene los mismos valores de *ids*, *cámara* y *vídeo*, pero valores diferentes de *hora*, *lat* y *long*. (Se han empleado la latitud y la longitud para denotar la posición.) La clave de esta tabla se puede representar en forma de dependencia funcional: $SHLN \rightarrow CV$, donde *N* significa longitud. Hay otra dependencia: $S \rightarrow CV$. Por tanto, esta relación no se halla en FNBC; de hecho, ni siquiera se encuentra en 3FN. Se puede descomponer Sondas para obtener un esquema en FNBC:

Posi_Sondas(*ids: integer, hora: timestamp, lat: real, long: real*)
Vídeo_Sondas(*ids: integer, cámara: string, vídeo: BLOB*)

Este diseño es prácticamente el mejor que se puede conseguir en un SGBDR. No obstante, presenta varios inconvenientes.

En primer lugar, la representación de vídeos como BLOB implica que hay que escribir código de aplicación en un lenguaje externo para manipular el objeto de vídeo de la base de datos. Considérese esta consulta: “Para la sonda 10, muéstrese el vídeo grabado entre la 1:10 P.M. y la 1:15 P.M. del 10 de mayo de 1996.” Hay que recuperar todo el objeto de vídeo asociado con la sonda 10, grabado a lo largo de varias horas, para mostrar un fragmento grabado en cinco minutos.

Además, el hecho de que cada sonda tenga una secuencia asociada de lecturas de posición queda oscurecido y la información de secuencias asociada con cada sonda se dispersa entre varias tuplas. Un tercer inconveniente es que resulta necesario separar la información de vídeo de la información de secuencias de cada sonda. Estas limitaciones quedan al descubierto mediante consultas que obligan a considerar toda la información asociada con cada sonda; por ejemplo, “Para cada sonda, imprímase el primer momento de grabación y el tipo de cámara.” Esta consulta supone ahora una reunión de Posi_Sondas y de Vídeo_Sondas sobre el campo *ids*.

Diseño de base de datos SGBDROO

Los SGBDROO suponen una solución mucho mejor. En primer lugar, se puede guardar el vídeo como objeto TAD y escribir métodos que capturen cualquier tratamiento especial que se desee llevar a cabo. En segundo lugar, dado que permiten guardar tipos estructurados como las listas, se puede guardar la secuencia de posiciones de cada sonda en una sola tupla, junto con la información de vídeo. Este formato elimina la necesidad de llevar a cabo reuniones en consultas que impliquen tanto a la secuencia como a la información en vídeo. El diseño SGBDROO para el ejemplo consta de una sola relación denominada `TodaInfo_Sondas`:

```
TodaInfo_Sondas(ids: integer, secpos: sec_posición, cámara: string,
vídeo: corriente_mpeg)
```

Esta definición supone dos tipos nuevos, `sec_posición` y `corriente_mpeg`. Este último tipo se define como TAD, con el método `mostrar()`, que toma un momento inicial y un momento final y muestra la parte de vídeo grabada durante ese intervalo. Este método se puede implementar de manera eficiente mediante el examen de la duración total de la grabación y de la longitud total del vídeo y la interpolación para extraer el fragmento grabado durante el intervalo especificado en la consulta.

La primera consulta en SQL ampliado que utiliza este método `mostrar` es la siguiente. En este caso sólo se recuperará el segmento de vídeo solicitado, en vez del vídeo entero.

```
SELECT mostrar(T.vídeo, 1:10 P.M. del 10 de mayo de 1996,
               1:15 P.M. del 10 de mayo de 1996)
FROM   TodaInfo_Sondas T
WHERE  T.ids = 10
```

Considérese ahora el tipo `sec_posición`. Se podría definir como tipo `lista`, que contiene una lista de los objetos del tipo `ROW`:

```
CREATE TYPE sec_posición
  (row (hora: timestamp, lat: real, long: real))
```

Considérese el campo `secpos` de cualquier fila para una sonda dada. Este campo contiene una lista de filas, cada una de las cuales tiene tres campos. Si el SGBDROO implementa los tipos colección en toda su generalidad, se debería poder extraer la columna `hora` de esta lista para obtener una lista de valores `timestamp` y aplicar el operador de agregación `MIN` a esta lista para averiguar el primer momento de grabación de la sonda dada. El soporte de los tipos colección permite expresar de esta manera la segunda consulta:

```
SELECT  T.ids, MIN(T.secpos.hora)
FROM    TodaInfo_Sondas T
```

Los SGBDROO actuales no son tan generales ni tan limpios como sugiere esta consulta. Por ejemplo, puede que el sistema no reconozca que la proyección de la columna `hora` desde una lista de filas proporciona una lista de valores de marcas temporales; o que sólo permita aplicar operadores de agregación a tablas y no a valores de listas anidadas.

Continuando con el ejemplo, puede que se desee llevar a cabo operaciones especializadas con las secuencias de posición que vayan más allá de los operadores de agregación habituales. Por ejemplo, puede que se desee definir un método que tome un intervalo de tiempo y

calcule la distancia recorrida por la sonda durante ese intervalo. El código para ese método debe comprender detalles de la trayectoria de las sondas y de los sistemas de coordenadas geoespaciales. Por esos motivos, puede que se decida definir `sec_posición` como TAD.

Evidentemente, un SGBDROO (ideal) ofrece muchas opciones de diseño útiles que no se hallan disponibles en los SGBDR.

15.7.2 Identidad de los objetos

A continuación se examinarán algunas de las consecuencias del empleo de tipos referencia o idos. El uso de idos resulta especialmente significativo cuando el tamaño del objeto es grande, bien debido a que es un tipo de datos estructurado, bien debido a que es un objeto de gran tamaño, como las imágenes.

Aunque los tipos referencia y los tipos estructurados parezcan similares, en realidad son bastante diferentes. Por ejemplo, considérese el tipo estructurado `mi_cine tuple(númc integer, nombre text, dirección text, teléfono text)` y el tipo referencia `ref cine(t_cine)` de la Figura 15.1. Existen diferencias importantes en el modo en que las actualizaciones de la base de datos afectan a estos dos tipos:

- **Eliminación.** Los objetos con referencias pueden verse afectados por la eliminación de objetos a los que hagan referencia, mientras que los objetos estructurados sin referencias no se ven afectados por la eliminación de otros objetos. Por ejemplo, si se eliminara de la base de datos la tabla Cines, los objetos del tipo `cine` podrían cambiar de valor a `null`, ya que los objetos `t_cine` a los que hacen referencia se habrían eliminado, mientras que los objetos similares del tipo `mi_cine` no cambiarían de valor.
- **Actualización.** Los objetos de los tipos referencia cambian de valor si se actualizan los objetos a los que hacen referencia. Los objetos de tipos estructurados sin referencias sólo cambian de valor si se actualizan directamente.
- **Compartir y copiar.** A cada objeto identificado le pueden hacer referencia varios elementos de tipos referencia, por lo que cada actualización del objeto queda reflejada en muchos sitios. Para obtener un efecto parecido en tipos sin referencias hace falta actualizar todas las “copias” del objeto.

También hay diferencias importantes en cuanto al almacenamiento entre los tipos referencia y los tipos sin referencias, lo que podría afectar al rendimiento:

- **Sobrecarga de almacenamiento.** Es posible que guardar copias de un valor grande en varios objetos de tipo estructurado ocupe mucho más espacio que guardar el valor una sola vez y hacer referencia a él en otros lugares mediante los objetos de tipo referencia. Esta necesidad adicional de almacenamiento puede afectar tanto al empleo del disco como a la gestión de las memorias intermedias (si se tiene acceso a muchas copias a la vez).
- **Agrupación.** Los integrantes de los objetos estructurados se suelen guardar juntos en disco. Puede que los objetos con referencias apunten a otros objetos que se hallen distantes en el disco, y que el brazo del disco necesite un movimiento significativo para componer el objeto con sus referencias. Por tanto, los objetos estructurados pueden ser más eficientes que los tipos referencia si normalmente se tiene acceso a ellos en su totalidad.

IDO e integridad referencial. En SQL:1999 se exige que todos los idos que aparecen en una columna de una relación hagan referencia a la misma relación objetivo. Este “enfoque” permite comprobar la “integridad referencial” de las referencias de los idos igual que se hace con las referencias de clave externa. Aunque los productos de SGBDROO actuales que soportan los idos no permiten estas comprobaciones, es probable que lo hagan en versiones futuras. Esto hará mucho más seguro el empleo de idos.

Muchos de estos problemas surgen también en los lenguajes de programación tradicionales como C o Pascal, que distinguen entre los conceptos de hacer referencia a los objetos *por su valor* o *por su referencia*. En el diseño de bases de datos la decisión de utilizar tipos estructurados o de referencia suele incluir la consideración de los costes de almacenamiento, de los problemas de agrupación y del efecto de las actualizaciones.

Identidad de los objetos y claves externas

El empleo de idos para hacer referencia a objetos es parecido al empleo de claves externas para hacer referencia a tuplas de otra relación, pero no exactamente igual: los idos pueden apuntar a objetos de `t_cine` que se guarden *en cualquier parte* de la base de datos, incluso en campos, mientras que las referencias de clave externa están limitadas a apuntar a objetos de la relación concreta a la que se hace referencia. Esta restricción permite que el SGBD ofrezca mucho más soporte para la integridad referencial que para los punteros ido arbitrarios. En general, si se elimina un objeto mientras sigue habiendo punteros ido que apunten a él, todo lo que puede hacer el SGBD es reconocer la situación mediante el mantenimiento de un contador de referencias. (Incluso este soporte limitado se hace imposible si se pueden copiar los idos libremente.) Por tanto, si los idos se utilizan para hacer referencia a objetos, la responsabilidad de que haya referencias “colgadas” recae principalmente sobre el usuario. Esta grave responsabilidad sugiere que los idos se deben emplear con gran cautela y que se deben utilizar en su lugar las claves externas siempre que resulte posible.

15.7.3 Ampliación del modelo ER

El modelo ER, tal y como se describe en el Capítulo 2, no resulta adecuado para el diseño de SGBDROO. Hay que utilizar un modelo ER ampliado que soporte los atributos estructurados (es decir, los conjuntos, las listas, los arrays como valores de los atributos), distinga si las entidades tienen identificadores de objeto y permita modelar entidades cuyos atributos incluyan métodos. Estos comentarios se ilustrarán mediante un diagrama ER ampliado para describir los datos de las sondas espaciales de la Figura 15.8; los convenios de la notación son *ad hoc* y sólo tienen finalidad ilustrativa.

La definición de sondas de la Figura 15.8 tiene dos aspectos nuevos. En primer lugar, tiene el atributo de tipo estructurado `listof(row(hora, lat, long))`; cada valor asignado a este atributo en una entidad Sondas es una lista de tuplas con tres campos. En segundo lugar, Sondas tiene un atributo denominado `vídeo` que es un objeto del tipo abstracto de datos, lo

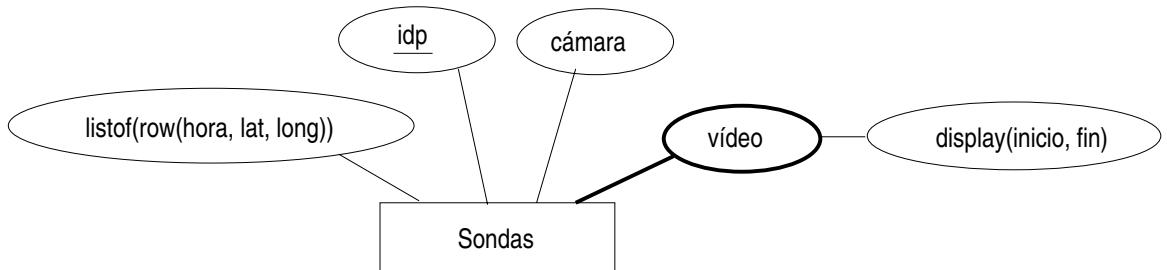


Figura 15.8 El conjunto de entidades Sondas

que se indica mediante un óvalo oscuro para este atributo con una línea oscura que lo une con Sondas. Además, este atributo tiene un “atributo” propio, que es un método del TAD.

De manera alternativa se podría modelar cada vídeo como entidad mediante un conjunto de entidades denominado Vídeos. La asociación entre las entidades Sondas y Vídeos se podría capturar así mediante la definición de un conjunto de relaciones que los vinculara. Dado que cada vídeo lo recibe exactamente una sonda y que todos los vídeos los recibe alguna sonda, esta relación se puede mantener guardando simplemente una referencia a un objeto sonda con cada entidad Vídeos; esta técnica es básicamente el segundo enfoque de traducción de los diagramas ER a tablas que se analiza en el Apartado 2.4.1.

Si también se hace de Vídeos un conjunto de entidades débiles en este diseño alternativo, se puede añadir una restricción de integridad referencial que haga que la entidad Vídeos se elimine cuando se elimine la entidad Sondas correspondiente. Este diseño alternativo ilustra con mayor generalidad un estrecho parecido entre las referencias de almacenamiento a objetos y las claves externas; el mecanismo de la clave externa consigue el mismo efecto que los idos de almacenamiento, pero de manera controlada. Si se emplean idos, el usuario debe garantizar que no haya referencias colgadas cuando se elimina un objeto, con muy poco soporte por parte del SGBD.

Finalmente, hay que destacar que se necesita una ampliación significativa del modelo ER para que soporte el diseño de colecciones anidadas. Por ejemplo, si se modela como entidad una secuencia de posiciones y se desea definir un atributo de Sondas que contenga un conjunto de esas entidades, no hay manera de hacerlo sin ampliar el modelo ER. Este punto no se tratará más al nivel de los diagramas ER, pero a continuación se considerará un ejemplo que ilustra cuándo se deben utilizar colecciones anidadas.

15.7.4 Empleo de colecciones anidadadas

Las colecciones anidadadas proporcionan una gran capacidad de modelado, pero también dan lugar a difíciles decisiones de diseño. Considérese la siguiente manera de modelar las secuencias de posiciones (aquí se omiten otras informaciones sobre las sondas para simplificar el estudio):

Sondas1(*ids: integer*, *secpos: sec_posición*)

Se trata de una buena elección si las consultas importantes de la carga de trabajo exigen que se examine la secuencia de posiciones de una sonda concreta, como ocurre en la consulta

“Para cada sonda, determíñese la primera vez que realizó una grabación y el tipo de cámara empleada.” Por otro lado, considérese una consulta que exige que examinemos todas las secuencias de posiciones: “Búsquese la primera grabación existente para $lat=5$, $long=90$.” Esta consulta se puede responder de manera más eficiente si se emplea el esquema siguiente:

Sondas2(*ids*: integer, *hora*: timestamp, *lat*: real, *long*: real)

Por tanto, la elección de esquema debe guiarse por la carga de trabajo esperada (como siempre). Como ejemplo adicional, considérese el esquema siguiente:

Puede_Impartir1(*ida*: integer, *profesores*: setof(*nif*: string), *sueldo*: integer)

Si hay que interpretar las tuplas de esta tabla como “La asignatura *ida* puede impartirla cualquiera de los profesores del campo *profesores*, con un coste de *sueldo*”, es posible emplear en su lugar el esquema siguiente:

Puede_Impartir2(*ida*: integer, *nif_profesor*: string, *sueldo*: integer)

Se puede optar entre estas dos alternativas en función del modo en que se espera consultar la tabla. Por otro lado, supóngase que las tuplas de Puede_impartir1 se deben interpretar como “La asignatura *ida* puede impartirla el equipo *profesores*, con un coste combinado de *sueldo*.” Puede_Impartir2 ya no constituye una alternativa viable. Si se deseara aplanar Puede_Impartir1, habría que utilizar una tabla diferente para codificar los equipos:

Puede_Impartir2(*ida*: integer, *id_equipo*: ido, *sueldo*: integer)

Equipos(*ide*: ido, *nif*: string)

Como muestran estos ejemplos, las colecciones anidadas resultan adecuados en ciertas situaciones, pero esta característica se puede emplear mal con facilidad; por tanto, las colecciones anidadas se deben utilizar con cuidado.

15.8 DESAFÍOS EN LA IMPLEMENTACIÓN DE LOS SGBDROO

La funcionalidad mejorada de los SGBDROO suscita varios desafíos en su implementación. Algunos de estos desafíos se comprenden bien y en los diferentes productos se han implementado soluciones; otros son objeto de investigación en este momento. En este apartado se examinarán algunos de los desafíos fundamentales que surgen al implementar un SGBDROO eficiente y completamente funcional, aunque existen muchos más aspectos implicados que los que aquí se analizan.

15.8.1 Almacenamiento y métodos de acceso

Dado que las bases de datos relacionales orientadas a objetos guardan nuevos tipos de datos, los implementadores de SGBDROO deben volver a examinar algunos de los aspectos de almacenamiento e indexación tratados en capítulos anteriores. En concreto, el sistema debe guardar de manera eficiente los objetos TAD y los objetos estructurados y ofrecer un acceso indexado eficiente a los dos.

Almacenamiento de objetos TAD y de tipo estructurado de gran tamaño

Los objetos TAD y estructurados de gran tamaño complican la disposición de los datos en el disco. Este problema se comprende bien y se ha resuelto prácticamente en todos los SGBDROO y SGBDOO. Aquí se presentarán algunos de sus aspectos principales.

Los TAD definidos por los usuarios pueden ser bastante grandes. En especial, pueden ser mayores que una página de disco. Los TAD de gran tamaño, como los BLOB, necesitan un almacenamiento especial, generalmente en una posición del disco diferente de las tuplas que los contienen. Se mantienen punteros basados en el disco desde las tuplas a los objetos que contienen.

Los objetos estructurados también pueden ser de gran tamaño pero, a diferencia de los objetos TAD, varían a menudo de tamaño durante la existencia de la base de datos. Por ejemplo, considérese el atributo *protagonistas* de la tabla *películas* de la Figura 15.1. A medida que pasan los años, puede que algunos de los “actores secundarios” de una película antigua se hagan famosos⁶. Puede que, cuando un actor secundario se haga famoso, Pequeño desee dar a conocer su presencia en películas anteriores. Esto supone una inserción en el atributo *protagonistas* de una tupla de *películas*. Debido a que estos atributos masivos pueden crecer de manera arbitraria se necesitan mecanismos dinámicos de formato de los discos.

Surge una complicación adicional con los tipos array. Tradicionalmente, los elementos de los arrays se guardan en disco secuencialmente fila por fila; por ejemplo,

$$A_{11}, \dots, A_{1n}, A_{21}, \dots, A_{2n}, \dots, A_{m1}, \dots, A_{mn}$$

Sin embargo, puede que las consultas soliciten a menudo subarrays que no se guarden en disco de manera contigua (por ejemplo, $A_{11}, A_{21}, \dots, A_{m1}$). Esas solicitudes pueden dar lugar a un coste de E/S muy elevado para la recuperación del subarray. Para reducir el número de operaciones de E/S se suele descomponer los arrays en *fragmentos* contiguos, que luego se guardan en el disco en un orden determinado. Aunque cada fragmento sea una región contigua del array, no necesitan ir fila por fila ni columna por columna. Por ejemplo, puede que un fragmento de tamaño 4 sea $A_{11}, A_{12}, A_{21}, A_{22}$, que es una región cuadrada si se piensa en el array como si estuviera organizado fila por fila en dos dimensiones.

Indexación de tipos nuevos

Un motivo importante de que los usuarios coloquen sus datos en bases de datos es permitir un acceso eficiente mediante los índices. Por desgracia, las estructuras de índices habituales de los SGBDR sólo soportan condiciones de igualdad (los árboles B+ y los índices de asociación) y de rango (los árboles B+). Un problema importante de los SGBDROO es que ofrecen índices eficientes para los métodos TAD y operadores para los objetos estructurados.

Los investigadores han propuesto muchas estructuras de índices especializadas para aplicaciones concretas como la cartografía, la investigación genómica, los repositorios multimedia, la búsqueda en Web, etcétera. Para una empresa de SGBDROO resulta imposible implementar todos los índices que se han inventado. En vez de eso, el usuario debe poder ampliar el

⁶Un ejemplo famoso es Marilyn Monroe, que tenía un papelito en el clásico *Eva al desnudo* con Bette Davis.

conjunto de estructuras de índices del SGBDROO. La posibilidad de ampliación permite que, por ejemplo, un experto en cartografía no sólo registre un TAD para los puntos de un mapa (es decir, pares latitud-longitud) sino que también implemente una estructura de índices que soporte consultas naturales de los mapas (por ejemplo, los árboles R, que se adaptan a condiciones como “Buscar todos los cines a menos de 200 kilómetros de Andorra”).

Una manera de hacer que el conjunto de estructuras de índices sea extensible es publicar una *interfaz de métodos de acceso* que permita que los usuarios implementen estructuras de índices *fuera* del SGBD. El índice y los datos se pueden guardar en un sistema de archivos y el SGBD se limita a formular las peticiones básicas sobre iteradores *abrir*, *siguiente* y *cerrar* al código del índice externo del usuario. Esta funcionalidad permite que un usuario conecte el SGBD a un motor de búsqueda Web, por ejemplo. Un inconveniente importante de este enfoque es que los datos del índice externo no están protegidos por el soporte del SGBD de la concurrencia y de la recuperación. Una alternativa es que el SGBDROO ofrezca una “plantilla” genérica para estructuras de índices que sea lo bastante general como para abarcar la mayor parte de las estructuras de índices que puedan inventar los usuarios. Como esa estructura se implementa en el SGBD, puede soportar la elevada concurrencia y la recuperación. Una de esas estructuras es el *árbol de búsqueda generalizado* (*Generalized Search Tree*) (GiST). Se trata de una plantilla para estructuras de índices basada en los árboles B+, que permite que se implemente la mayor parte de las estructuras de índices inventadas hasta ahora con tan sólo unas cuantas líneas de código de TAD definido por el usuario.

15.8.2 Procesamiento de consultas

Los TAD y los tipos estructurados necesitan una nueva funcionalidad para el procesamiento de las consultas en los SGBDROO. También modifican varias suposiciones que afectan a la eficiencia de esas consultas. En este apartado se examinarán dos problemas de funcionalidad (las funciones de agregación definidos por los usuarios y la seguridad) y dos de eficiencia (el almacenamiento de métodos en la memoria intermedia y el rescate de punteros).

Funciones de agregación definidas por los usuarios

Dado que se permite que los usuarios definan métodos nuevos para sus TAD, no resulta irracional esperar que también deseen definir nuevas funciones de agregación para sus TAD. Por ejemplo, las funciones de agregación habituales de SQL —COUNT, SUM, MIN, MAX, AVG— no resultan especialmente apropiadas para el tipo *imagen* del esquema Pequeño.

La mayor parte de los SGBDROO permiten que los usuarios registren en el sistema funciones de agregación nuevas. Para registrar una función de agregación el usuario debe implementar tres métodos, denominados *inicializar*, *iterar* y *terminar*. El método *inicializar* inicializa el estado interno para la agregación. El método *iterar* actualiza ese estado para todas las tuplas vistas, mientras que el método *terminar* calcula el resultado de la agregación con base en el estado final y luego limpia todo. Por ejemplo, considérese una función de agregación que calcule el segundo valor más elevado de un campo. La llamada *inicializar* asignaría espacio de almacenamiento para los dos valores más elevados, la llamada *iterar* compararía el valor de la tupla actual con los dos más elevados y los actualizaría si fuera necesario, mientras que la

llamada *terminar* eliminaría el espacio de almacenamiento para los dos valores más elevados y devolvería una copia del segundo de ellos.

Seguridad para los métodos

Los TAD ofrecen a los usuarios la posibilidad de añadir código al SGBD; de esta posibilidad se puede abusar. Un método TAD defectuoso o malévolamente escrito puede hacer que se caiga el servidor de la base de datos o, incluso, que se corrompa ésta. El SGBD debe tener mecanismos para evitar que el código defectuoso o malévolamente escrito cause problemas. Puede que tenga sentido desactivar estos mecanismos en aras de la eficiencia en entornos de producción con métodos proporcionados por el fabricante. No obstante, es importante que esos mecanismos existan, aunque sólo sea para que soporten la depuración de los métodos TAD; en caso contrario, los escritores de métodos deberían escribir código sin fallos antes de registrar sus métodos en el SGBD —lo que no es un entorno de programación muy tolerante—.

Un mecanismo para evitar problemas es hacer que los métodos de usuario se *interpreten* en lugar de *compilarse*. El SGBD puede comprobar si el método se comporta bien restringiendo la capacidad del lenguaje interpretado o asegurándose de que cada paso dado por el método sea seguro antes de ejecutarlo. Entre los lenguajes interpretados habituales para esta finalidad están Java y las partes procedimentales de SQL:1999.

Un mecanismo alternativo es permitir que los métodos de usuario se compilen desde un lenguaje de programación de propósito general, como C++, pero ejecutar esos métodos en un espacio de direcciones diferente del SGBD. En ese caso, el SGBD envía comunicaciones entre procesos (interprocess communications, IPC) explícitas al método de usuario, que las devuelve a su vez. Este enfoque evita que los fallos de los métodos de usuario (por ejemplo, punteros mal dirigidos) corrompan el estado del SGBD o de la base de datos e impide que los métodos malévolos lean o modifiquen también el estado del SGBD o la base de datos. Obsérvese que no hace falta que el usuario que escribe el método sepa que el SGBD lo ejecuta en un proceso separado: el código de usuario se puede vincular con un “envoltorio” que transforme las llamadas al método y los valores devueltos en IPC.

Almacenamiento de métodos en memoria intermedia

Los métodos TAD definidos por el usuario pueden resultar muy costosos de ejecutar y suponer la mayor parte del tiempo empleado en procesar las consultas. Durante el procesamiento de consultas puede que tenga sentido guardar en la memoria intermedia el resultado de los métodos, por si se los llamará varias veces con el mismo argumento. En el ámbito de una sola consulta se puede evitar llamar dos veces al mismo método para los valores duplicados de una columna ordenando la tabla de acuerdo con esa columna o empleando un esquema basado en la asociación muy parecido al utilizado para la agregación. Una alternativa es mantener una *memoria intermedia* de las entradas de los métodos y de sus resultados correspondientes en forma de tabla de la base de datos. Así, para hallar el valor de un método para unas entradas concretas basta prácticamente con reunir las tablas de entrada con la tabla de memoria intermedia. También se pueden combinar estos dos enfoques.

Rescate de punteros

En algunas aplicaciones los objetos se recuperan en la memoria y se tiene frecuente acceso a ellos mediante sus idos; su desreferencia se debe llevar a cabo de manera muy eficiente. Algunos sistemas conservan una tabla de idos de los objetos que se hallan (en ese momento) en la memoria. Cuando el objeto O se lleva a memoria, se comprueba cada ido contenido en O y se sustituyen los idos de los objetos presentes en memoria por punteros en memoria a esos objetos. Esta técnica, denominada **rescate de punteros**, hace referencias muy rápidas a los objetos que se hallan en la memoria. El inconveniente es que cuando un objeto sale de la página de memoria, las referencias a él en memoria se deben invalidar de alguna manera y sustituirse por su ido.

15.8.3 Optimización de consultas

Los índices y técnicas de procesamiento de consultas nuevos amplían las opciones disponibles para los optimizadores de consultas. Para tratar la nueva funcionalidad de procesamiento de consultas el optimizador debe conocer la nueva funcionalidad y emplearla de manera adecuada. En este apartado se examinarán dos aspectos de la exposición de información al optimizador (índices nuevos y estimación de los métodos TAD) y un aspecto de la planificación de consultas que se ignoró en los sistemas relacionales (la optimización de las selecciones costosas).

Registro de índices en el optimizador

A medida que se añaden estructuras de índices nuevas al sistema —bien mediante interfaces externas, bien mediante estructuras de plantillas incorporadas como los GiST— hay que informar al optimizador de su existencia y de sus costes de acceso. En concreto, para una estructura de índices dada, el optimizador debe conocer (1) a qué condiciones de cláusulas WHERE responde ese índice y (2) cuál es el coste de capturar una tupla para ese índice. Dada esta información, el optimizador puede emplear cualquier estructura de índices para crear el plan de consultas. Los diferentes SGBDROO se diferencian en la sintaxis del registro de las nuevas estructuras de índices. La mayor parte de los sistemas exigen que los usuarios declaren un número que represente el coste de acceso, pero una alternativa es que el SGBD mida la estructura cuando se utilice y mantenga una estadística actualizada del coste.

Factor de reducción y estimación de costes para los métodos TAD

El **factor de reducción** es la proporción entre el tamaño esperado del resultado y el tamaño total del origen de datos en una selección. Se puede estimar el factor de reducción de diferentes condiciones de selección y de reunión con condiciones como $=$ y $<$. Para las condiciones definidas por el usuario como `es_heriberto()` el optimizador también necesita poder estimar los factores de reducción. La estimación de los factores de reducción para las condiciones definidas por los usuarios es un problema difícil y se estudia activamente. El enfoque más

Posibilidad de ampliación del optimizador. Por ejemplo, considérese el optimizador de Oracle 9i, que se puede ampliar y soporta índices y métodos sobre “dominios”. El soporte incluye estadísticas y funciones de coste definidas por los usuarios que el optimizador emplea junto con las estadísticas del sistema. Supóngase que hay un índice de dominio para el texto de la columna *reanudar* y un índice normal de árboles B de Oracle para *fechacontrato*. Una consulta con una selección para estos dos campos se puede evaluar mediante la conversión de los idrs de los dos índices en mapas de bits, la realización de un AND para los mapas de bits y la conversión del mapa de bits en idrs antes del acceso a la tabla. Por supuesto, el optimizador considera también el empleo de los dos índices por separado, así como de una exploración de toda la tabla.

popular actualmente es dejarla en manos del usuario —los usuarios que registran métodos también pueden registrar una función auxiliar que estime el factor de reducción de esos métodos—. Si no se registra esa función, el optimizador emplea un valor arbitrario como $\frac{1}{10}$.

Los métodos TAD pueden resultar bastante costosos y es importante que el optimizador sepa exactamente lo que cuesta ejecutarlos. Una vez más, la estimación del coste de los métodos sigue siendo objeto de investigación. En los sistemas actuales los usuarios que registran un método pueden especificar su coste en forma de número, generalmente en unidades del coste de una operación de E/S del sistema. Es difícil que los usuarios realicen esta estimación de forma precisa. Una alternativa atractiva es que el SGBDROO ejecute el método para objetos de diferentes tamaños e intente estimar su coste de manera automática, pero este enfoque no se ha investigado con detalle y no se implementa en los SGBDROO comerciales.

Optimización de selecciones costosas

En los sistemas relacionales se espera que la selección sea una operación de tiempo nulo. Por ejemplo, no necesita operaciones de E/S y sólo unos cuantos ciclos de CPU para comprobar si *emp.sueldo < 10*. Sin embargo, las condiciones como *es_heriberto(Fotogramas.imagen)* pueden resultar bastante costosas, ya que pueden capturar objetos de gran tamaño fuera del disco y procesarlas en memoria de manera complicada.

Los optimizadores de SGBDROO deben considerar detenidamente la manera de ordenar las condiciones de selección. Por ejemplo, considérese una consulta de selección que pruebe las tuplas de la tabla Fotogramas con dos condiciones: *Fotogramas.númfotograma < 100* \wedge *es_heriberto(Fotograma.imagen)*. Probablemente sea preferible comprobar la condición para *númfotograma* antes que *es_heriberto*. La primera condición es rápida y puede que resulte falsa a menudo, lo que ahorra la molestia de comprobar la segunda condición. En general, la mejor ordenación de las selecciones es función de sus costes y de sus factores de reducción. Se puede demostrar que las selecciones se deben ordenar según su *rango* creciente, donde *rango* = (factor de reducción – 1)/coste. Si una selección con rango muy elevado aparece en una consulta multitabla, puede que incluso tenga sentido posponerla hasta que se hayan llevado a cabo las reuniones. Los detalles de la colocación óptima de las selecciones costosas entre las reuniones resultan algo complicados, lo que se añade a la complejidad de la optimización en los SGBDROO.

15.9 SGBDOO

En la introducción de este capítulo se definieron los SGBDOO como lenguajes de programación con soporte para los objetos persistentes. Aunque esta definición refleja con precisión los orígenes de los SGBDOO y, hasta cierto punto, el objetivo de la implementación de los SGBDOO, el hecho de que los SGBDOO soporten los *tipos colección* (véase el Apartado 15.2.1) permite ofrecer un lenguaje de consultas para los conjuntos. En realidad, el Grupo de Gestión de Bases de Datos de Objetos (Object Database Management Group, ODMG) ha desarrollado una norma denominada **lenguaje de consultas para objetos** (Object Query Language, **OQL**).

OQL se parece a SQL, con una sintaxis del tipo **SELECT- FROM- WHERE** (incluso se soportan **GROUP BY**, **HAVING** y **ORDER BY**) y muchas de las extensiones propuestas para SQL:1999. Es de destacar que OQL soporta los tipos estructurados, incluidos los conjuntos, las bolsas, los arrays y las listas. El tratamiento de los conjuntos por OQL es más uniforme que el de SQL:1999, en el sentido de que no da un tratamiento especial a los conjuntos de filas; por ejemplo, OQL permite que se aplique la operación de agregación **COUNT** a una lista para que calcule la longitud de la misma. OQL soporta también los tipos referencia, las expresiones de ruta, los TAD y la herencia, las ampliaciones de los tipos y las consultas anidadas al estilo de SQL. También hay un lenguaje de definición de datos normalizado para los SGBDOO (**lenguaje de datos para objetos** (Object Data Language, **ODL**)) que es parecido al subconjunto LDD de SQL pero soporta las características adicionales de los SGBDOO, como las definiciones de TAD.

15.9.1 El modelo de datos ODMG y ODL

El modelo de datos ODMG es la base de los SGBDOO, al igual que el modelo de datos relacional es la base de los SGBDR. Las bases de datos contienen conjuntos de **objetos**, que son parecidos a las entidades del modelo ER. Cada objeto tiene un ido único, y la base de datos contiene conjuntos de objetos con propiedades parecidas; este tipo de conjunto se denomina **clase**.

Las propiedades de cada clase se especifican mediante el ODL y son de tres tipos: atributos, relaciones y métodos. Los **atributos** tienen un tipo atómico o uno estructurado. ODL soporta los constructores de tipos **set**, **bag**, **list**, **array** y **struct**; se trata precisamente de **setof**, **bagof**, **listof**, **ARRAY** y **ROW** en la terminología del Apartado 15.2.1.

Las **relaciones** tienen un tipo que es una referencia a un objeto o un conjunto de esas referencias. Una relación captura el modo en que el objeto está relacionado con uno o varios objetos de la misma clase o de otra diferente. Las relaciones del modelo ODMG no son realmente más que relaciones binarias en el sentido del modelo ER. Cada relación tiene su **relación inversa** correspondiente; de manera intuitiva, se trata de la relación “en sentido contrario”. Por ejemplo, si una película se exhibe en varios cines y cada cine exhibe varias películas, hay dos relaciones inversas entre sí: *exhibidaEn* se asocia con la clase de las películas y es el conjunto de cines en el que se exhibe una película dada, mientras que *enCartel* se asocia con la clase de los cines y es el conjunto de películas que se exhiben en ese cine.

Los **métodos** son funciones que se pueden aplicar a los objetos de la clase. No hay ninguna analogía para los métodos en el modelo ER ni en el relacional.

Clase = Interfaz + Implementación. Hablando con propiedad, una clase consiste en una interfaz junto con una implementación de esa interfaz. La definición de interfaz de un ODL se implementa en un SGBDOO mediante su traducción en declaraciones del lenguaje orientado a objetos (por ejemplo, C++, Smalltalk o Java) soportado por el SGBDOO. Si, por ejemplo, se considera C++, hay una biblioteca de clases que implementa las estructuras del ODL. También hay un **lenguaje de manipulación de objetos** (Object Manipulation Language, **OML**) específico para ese lenguaje de programación (en este caso, C++), que especifica la manera en que se manipulan los objetos en el lenguaje de programación. El objetivo es integrar completamente el lenguaje de programación y las características de la base de datos.

La palabra clave **interface** se utiliza para definir una clase. Para cada interfaz se puede declarar una **extensión**, que es el nombre del conjunto actual de objetos de esa clase. La extensión es análoga a la instancia de las relaciones y la interfaz es análoga al esquema. Si el usuario no prevé la necesidad de trabajar con el conjunto de objetos de una clase dada —basta con manipular los objetos por separado— se puede omitir la declaración de extensión.

Las siguientes definiciones del ODL para las clases Película y Cine ilustran estos conceptos. (Aunque estas clases tengan cierto parecido con el esquema de la base de datos Pequeño, no se debería buscar un paralelismo exacto, ya que se ha modificado el ejemplo para destacar las características del ODL.)

```
interface Película
  (extent Películas key nombrePelícula)
  { attribute date estreno;
    attribute date retirada;
    attribute string nombrepelícula;
    relationship Set<Cine> exhibidaEn inverse Cine::enCartel;
  }
```

El conjunto de objetos de la base de datos cuya clase es Película se denomina Películas. No hay dos objetos de Películas que tengan el mismo valor de *nombrePelícula*, como indica la declaración de clave. Cada película se exhibe en un conjunto de cines durante el periodo especificado. (Sería más realista asociar un periodo diferente con cada cine, ya que normalmente cada película se exhibe en cines diferentes a lo largo de periodos de tiempo distintos. Aunque se puede definir una clase que capture este detalle, se ha escogido esta definición más sencilla para nuestro análisis.) Cada cine es un objeto de la clase Cine, definida como:

```
interface Cine
  (extent Cines key nombreCine)
  { attribute string nombreCine;
    attribute string dirección;
    attribute integer precioEntrada;
    relationship Set<Película> enCartel inverse Película::exhibidaEn;
    float númpelículas() raises(errorContandoPelículas);
  }
```

Cada cine exhibe varias películas y cobra el mismo precio de entrada para cada película. Obsérvese que se declara que las relaciones *exhibidaEn* de Película y *enCartel* de Cine son inversas entre sí. Cine también tiene el método *númpelículas()* que se puede aplicar a objetos cine para averiguar el número de películas que se exhiben en ellos.

El ODL también permite especificar jerarquías de herencias, como ilustra la siguiente definición de clase:

```
interface GalaEspecial extends Película
  (extent GalasEspeciales)
  { attribute integer aforoMáximo;
    attribute string instituciónBenéfica;
  }
```

Los objetos de la clase *GalaEspecial* son objetos de la clase *Película* con algunas propiedades adicionales, como se analiza en el Apartado 15.5.

15.9.2 OQL

El lenguaje de consulta OQL del ODMG se diseñó de manera deliberada para que tuviera una sintaxis similar a la de SQL para facilitar a los usuarios familiarizados con SQL el aprendizaje de OQL. Considérese la siguiente consulta que busca parejas de películas y cines tales que la película se exhiba en el teatro y el cine exhiba más de una película:

```
SELECT nombrep: P.nombrePelícula, nombrec: C.nombreCine
FROM Películas P, P.exhibidaEn C
WHERE C.númpelículas() > 1
```

La cláusula **SELECT** indica el modo en que se puede poner nombre a los campos del resultado: los dos campos del resultado se llaman *nombrep* y *nombrec*. La parte de esta consulta que se diferencia de SQL es la cláusula **FROM**. La variable *P* se vincula por turno a cada película de la extensión *Películas*. Para cada película dada *P* se vincula por turno la variable *C* a cada cine del conjunto *P.exhibidaEn*. Por tanto, el empleo de la expresión de ruta *P.exhibidaEn* permite expresar con facilidad una consulta anidada. La consulta siguiente ilustra la estructura de agrupación en OQL:

```
SELECT C.precioEntrada,
númProm: AVG(SELECT P.C.númpelículas() FROM partition P)
FROM Cines C
GROUP BY C.precioEntrada
```

Para cada valor del precio de la entrada se crea un grupo de cines con ese mismo precio de entrada. Ese grupo de cines es la partición para ese precio de la entrada, a la que se hace referencia mediante la palabra clave de OQL **partition**. En la cláusula **SELECT** se calcula, para cada precio de la entrada, el número medio de películas exhibidas en los cines de la partición para ese valor de *precioEntrada*. OQL soporta una variación interesante de la operación de agrupación de la que carece SQL:

```
SELECT bajo, alto,
```

```

númProm: AVG(SELECT P.C.númpelículas() FROM partition P)
FROM      Cines C
GROUP BY bajo: C.precioEntrada < 5, alto: C.precioEntrada >= 5

```

La cláusula GROUP BY crea ahora sólo las dos particiones denominadas *bajo* y *alto*. Cada objeto cine *C* se sitúa en una de esas particiones según el precio de sus entradas. En la cláusula SELECT *bajo* y *alto* son variables booleanas, una (y sólo una) de las cuales es siempre verdadera en cualquier tupla de resultados dada; partition se materializa para la partición correspondiente de los objetos cine. En el ejemplo se obtienen dos tuplas de resultados. Una de ellas tiene *bajo* igual a **verdadero** y *númProm* igual al número promedio de películas exhibidas en los cines con un precio de entrada bajo. La segunda tupla tiene *alto* igual a **verdadero** y *númProm* igual al número promedio de películas exhibidas en los cines con precio de entrada elevado.

La consulta siguiente ilustra el soporte de OQL a las consultas que devuelven otras colecciones aparte de conjuntos y multiconjuntos:

```

(SELECT  C.nombreCine
FROM    Cines C
ORDER BY C.precioEntrada DESC) [0:4]

```

La cláusula ORDER BY hace que el resultado sea una lista de nombres de cines ordenados por el precio de su entrada. Se puede hacer referencia a los elementos de una lista por su posición, comenzando con la posición 0. Por tanto, la expresión [0:4] extrae una lista que contiene el nombre de los cinco cines con precio de entrada más elevado.

OQL soporta también DISTINCT, HAVING, el anidamiento explícito de las subconsultas, las definiciones de vistas y otras características de SQL.

15.10 COMPARACIÓN ENTRE SGBDR, SGBDOO Y SGBDROO

Ahora que se han tratado las principales ampliaciones de los SGBD orientadas a objetos ha llegado el momento de considerar las dos variantes principales de las bases de datos orientadas a objetos, los SGBDOO y los SGBDROO, y compararlas con los SGBDR. Aunque ya se han presentado los conceptos subyacentes a las bases de datos orientadas a objetos, sigue siendo necesario definir los términos SGBDOO y SGBDROO.

Un **SGBDROO** es un SGBD relacional con las ampliaciones examinadas en este capítulo. (No todos los sistemas de SGBDROO soportan todas las ampliaciones de la forma general en que las hemos estudiado, pero el objetivo de este apartado es el propio paradigma más que los sistemas concretos.) Un **SGBDOO** es un lenguaje de programación con un sistema de tipos que soporta las características examinadas en este capítulo y permite que cualquier objeto de datos sea **persistente**; es decir, que sobreviva de una ejecución del programa a otra. Muchos sistemas actuales no se ajustan por completo a ninguna de estas definiciones, sino que se hallan mucho más cerca de una de las dos y se pueden clasificar en consecuencia.

15.10.1 SGBDR y SGBDROO

La comparación entre un SGBDR y un SGBDROO es sencilla. El SGBDR no soporta las ampliaciones estudiadas en este capítulo. La simplicidad resultante del modelo de datos facilita la optimización de las consultas en aras, por ejemplo, de una ejecución eficiente. El sistema relacional también resulta más sencillo de utilizar, ya que hay menos características que dominar. Por otro lado, es menos versátil que el SGBDROO.

15.10.2 SGBDOO y SGBDROO: similitudes

Tanto los SGBDOO como los SGBDROO soportan los TAD definidos por los usuarios, los tipos estructurados, la identidad de los objetos y los tipos referencia y la herencia. Ambos soportan un lenguaje de consultas para la manipulación de los tipos colección. Los SGBDROO soportan una forma ampliada de SQL, mientras que los SGBDOO soportan ODL/OQL. Las similitudes no son en modo alguno accidentales: los SGBDROO intentan de manera consciente añadir características de los SGBDOO a los SGBDR, y los SGBDOO, por su parte, han desarrollado lenguajes de consulta basados en los lenguajes de consulta relacionales. Tanto los SGBDOO como los SGBDROO ofrecen funcionalidad de SGBD como el control de la concurrencia y la recuperación.

15.10.3 SGBDOO y SGBDROO: diferencias

La diferencia fundamental es realmente una filosofía que se aplica hasta el final: los SGBDOO intentan añadir funcionalidad de SGBD a un lenguaje de programación, mientras que los SGBDROO intentan añadir tipos de datos más ricos a los SGBD relacionales. Aunque los dos tipos de bases de datos orientadas a objetos vayan convergiendo en términos de funcionalidad, esa diferencia en su filosofía subyacente (y, para la mayor parte de los sistemas, el enfoque de su implementación) tiene repercusiones importantes en términos de los aspectos a los que se da más importancia en el diseño de los SGBD y de la eficiencia con que se soportan las diferentes características, como indica la comparación siguiente:

- Los SGBDOO intentan conseguir una integración completa con un lenguaje de programación como C++, Java o Smalltalk. Esa integración no es un objetivo importante para los SGBDROO. SQL:1999, al igual que SQL-92, permite incorporar órdenes de SQL en un lenguaje anfitrión, pero la interfaz resulta muy evidente para el programador de SQL. (SQL:1999 proporciona también sus propias estructuras para lenguajes de programación ampliadas, como se vio en el Capítulo 6.)
- Los SGBDOO están pensados para aplicaciones en las que resulta adecuado un punto de vista centrado en los objetos; es decir, las sesiones de usuario típicas consisten en la recuperación de unos cuantos objetos y en trabajar con ellos durante largos períodos de tiempo, capturando de manera ocasional objetos relacionados (por ejemplo, objetos a los que hagan referencia los objetos originales). Puede que los objetos sean extremadamente grandes y haya que capturarlos a trozos; por tanto, se debe prestar atención al almacenamiento en la memoria intermedia de fragmentos de los objetos. Se espera que la mayor parte de las aplicaciones pueda guardar en la memoria intermedia los objetos

que necesiten en memoria, una vez recuperados del disco. Por tanto, se presta una atención considerable a hacer que las referencias a los objetos presentes en la memoria sean eficientes. Es probable que las transacciones sean de duración muy larga y que el mantenimiento de bloqueos hasta el final de cada transacción provoque un bajo rendimiento; por tanto, se deben emplear alternativas a los bloqueos de dos fases.

Los SGBDROO se optimizan para las aplicaciones en las que la atención se centra en grandes conjuntos de datos, aunque los objetos puedan tener estructuras complejas y ser de tamaño apreciable. Se espera que las aplicaciones recuperen los datos del disco de manera extensiva y la optimización del acceso a disco sigue siendo la principal preocupación en cuanto a la ejecución eficiente. Se supone que las transacciones son relativamente cortas y se suelen emplear técnicas tradicionales de los SGBDR para el control de concurrencia y para la recuperación.

- Las facilidades para las consultas de OQL no se soportan de manera eficiente en la mayor parte de los SGBDOO, mientras que son la pieza clave de los SGBDRO. De alguna manera esta situación es el resultado de las diferentes concentraciones de esfuerzo en el desarrollo de estos sistemas. En cierto modo, también es consecuencia de que los sistemas se optimizan para tipos de aplicaciones muy diferentes.

15.11 PREGUNTAS DE REPASO

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

- Considérese el ejemplo Pequeño ampliado del Apartado 15.1. Explíquese la manera en que justifica la necesidad de las siguientes características de las bases de datos orientadas a objetos: los *tipos estructurados definidos por los usuarios*, los *tipos abstractos de datos (TAD)*, la *herencia* y la *identidad de los objetos*. (**Apartado 15.1**)
- ¿Qué son los *tipos de datos estructurados*? ¿Qué son, en concreto, los *tipos colección*? Examíñese el grado hasta el que estos conceptos están soportados en SQL:1999. ¿Qué constructores de tipos importantes faltan? ¿Cuáles son las limitaciones de los constructores ROW y ARRAY? (**Apartado 15.2**)
- ¿Qué tipos de operaciones se deberían ofrecer para cada uno de los tipos de datos estructurados? ¿Hasta qué grado se incluye ese soporte en SQL:1999? (**Apartado 15.3**)
- ¿Qué es un *tipo abstracto de datos*? ¿Cómo se definen los métodos de los tipos abstractos de datos en los lenguajes de programación externos? (**Apartado 15.4**)
- Explíquese la *herencia* y el modo en que los tipos nuevos (denominados *subtipos*) amplían los tipos existentes (denominados *supertipos*). ¿Qué son la *sobrecarga de métodos* y la *vinculación tardía*? ¿Qué es una *jerarquía de colecciones*? Compárese con la herencia en los lenguajes de programación. (**Apartado 15.5**)
- ¿En qué se diferencian los *identificadores de objetos (idos)* de los identificadores de registros de los SGBD relacionales? ¿En qué se diferencian de los URL? ¿Qué es un *tipo referencia*? Defínanse igualdad *profunda* y *superficial* e ilústrense mediante un ejemplo. (**Apartado 15.6**)

- La multitud de tipos de datos de los SGBDROO permiten diseñar esquemas de bases de datos más naturales y eficientes, pero introducen algunas opciones de diseño nuevas. Examínense los problemas de diseño de las bases de datos SGBDROO e ilústrese el análisis mediante una aplicación de ejemplo. (**Apartado 15.7**)
- La implementación de un SGBDROO plantea nuevos retos. El sistema debe guardar TAD de gran tamaño y tipos estructurados que pudieran ser muy grandes. Se deben facilitar mecanismos de índices eficientes y extensibles. Entre los ejemplos de funcionalidades nuevas están las *funciones de agregación definidas por los usuarios* (se pueden definir funciones de agregación nuevas para nuestros TAD) y la *seguridad de los métodos* (el sistema tiene que impedir que los métodos definidos por los usuarios pongan en peligro la seguridad del SGBD). Entre los ejemplos de técnicas nuevas para el aumento del rendimiento están el *almacenamiento de métodos en memoria intermedia* y el *rescate de punteros*. El optimizador debe conocer las funcionalidades nuevas y emplearlas de manera adecuada. Ilústrese cada uno de estos retos con un ejemplo. (**Apartado 15.8**)
- Compárense los SGBDOO con los SGBDROO. En especial, compárense OQL y SQL:1999 y analícese el modelo de datos subyacente. (**Apartados 15.9 y 15.10**)

EJERCICIOS

Ejercicio 15.1 Respóndase brevemente las preguntas siguientes:

1. ¿Cuáles son las nuevas clases de tipos de datos soportados en los sistemas de bases de datos orientados a objetos? Dese un ejemplo de cada una de ellas y examínese el modo en que se manejaría la situación del ejemplo si sólo se dispusiera de un SGBDR.
2. ¿Qué deben hacer los usuarios para definir TAD nuevos?
3. Permitir que los usuarios definan métodos puede conducir a mejoras en la eficiencia. Dese un ejemplo.
4. ¿Qué es la vinculación tardía de los métodos? Dese un ejemplo de herencia que ilustre la necesidad de la vinculación dinámica.
5. ¿Qué son las jerarquías de colecciones? Dese un ejemplo que ilustre el modo en que las jerarquías de colecciones facilitan las consultas.
6. Examínese el modo en que los SGBD aprovechan la encapsulación para implementar el soporte de los TAD.
7. Dese un ejemplo que ilustre las operaciones de anidamiento y desanidamiento.
8. Describanse dos objetos que sean iguales en profundidad pero no iguales superficialmente o explíquese el motivo de que esto no sea posible.
9. Describanse dos objetos que sean iguales superficialmente pero no iguales en profundidad o explíquese el motivo de que esto no sea posible.
10. Compárense los SGBDR y los SGBDROO. Describábase una situación de una aplicación para la que elegiría un SGBDR y explíquese el motivo. De manera análoga, describábase una situación de una aplicación para la que elegiría un SGBDROO y explíquese el motivo.

Ejercicio 15.2 Considérese el esquema Pequeño de la Figura 15.1 y todos los métodos relacionados definidos en este Capítulo. Escríbanse las consultas siguientes en SQL:1999:

1. ¿Cuántas películas se exhibieron en el cine $tno = 5$ entre el 1 de enero y el 1 de febrero de 2002?
2. ¿Cuál es el presupuesto más bajo de una película con, al menos, dos protagonistas?

3. Considérense los cines en los que se estrenó una película dirigida por Steven Spielberg el 1 de enero de 2002. Para cada uno de esos cines, imprima el nombre de todos los condados que se hallen a menos de 200 kilómetros. (Se pueden utilizar los métodos *solapamiento* y *radio* ilustrados en la Figura 15.2.)

Ejercicio 15.3 En la base de datos de una empresa hay que guardar información sobre los empleados, los departamentos y los hijos de los empleados. Para cada empleado, identificado por su *nif*, hay que registrar la información de *años* (el número de años que el empleado ha trabajado para la empresa) *teléfono* y *foto*. Hay dos subclases de empleados: temporales y fijos. El sueldo se calcula invocando un método que toma *años* como parámetro; este método tiene una implementación diferente para cada subclase. Además, para cada empleado fijo hay que registrar el nombre y la edad de cada uno de sus hijos. Las consultas que implican a los hijos más frecuentes son parecidas a “Buscar la edad promedio de los hijos de Benito” e “Imprimir el nombre de todos los hijos de Benito”.

Las fotos son objetos de imagen de gran tamaño y se pueden guardar en varios formatos de imagen (por ejemplo, gif o jpeg). Se desea definir un método *mostrar* para los objetos de imagen; mostrar se debe definir de manera diferente para cada formato de imagen. Para cada departamento, identificado por *númd*, hay que registrar la información de *nombred*, *presupuesto* y *trabajadores*. *Trabajadores* es el conjunto de empleados que trabajan en un departamento dado. Entre las consultas que implican trabajadores habituales están “Buscar el sueldo medio de todos los trabajadores (de todos los departamentos)”.

1. Empleando SQL ampliado, diseñese el esquema de un SGBDROO para la base de datos de la empresa. Muéstrense todas las definiciones de tipos, incluidas las definiciones de los métodos.
2. Si hay que guardar esta información en un SGBDR, ¿cuál el mejor diseño posible?
3. Compárense el diseño del SGBDROO y el del SGBDR.
4. Si se le comunica que una petición frecuente es mostrar las imágenes de todos los empleados de un departamento dado, ¿cómo emplearía esa información para el diseño físico de la base de datos?
5. Si se le comunica que hay que mostrar la imagen de los empleados siempre que se recupere alguna información sobre ellos, ¿afectaría eso al diseño de su esquema?
6. Si se le comunica que una consulta frecuente es buscar todos los empleados que son parecidos a una cierta imagen y se le da código que le permite crear un índice para todas las imágenes para que soporte la recuperación de imágenes parecidas, ¿qué haría para aprovechar ese código en un SGBDROO?

Ejercicio 15.4 Los SGBDROO deben soportar el acceso eficiente a través de las jerarquías de colecciones. Considérese la jerarquía de colecciones de los Cines y de los Cine_cafés presentada en el ejemplo Pequeño. En su papel de implementador del SGBD (no como ABD) debe evaluar tres alternativas de almacenamiento para estas tuplas:

- Todas las tuplas de todos los tipos de cines se guardan juntas en disco en un orden arbitrario.
 - Todas las tuplas de todos los tipos de cines se guardan juntas en disco, con las tuplas que son de Cine_cafés guardadas directamente después de la última de las tuplas que corresponden a cines que no son cafés.
 - Las tuplas de los Cine_cafés se guardan por separado del resto de las tuplas de los cines (que no son cafés).
1. Para cada opción de almacenamiento describese un mecanismo para distinguir entre las tuplas de cines normales y las de los Cine_cafés.
 2. Para cada opción de almacenamiento describese el modo de manejar la inserción de nuevas tuplas de cines que no son cafés.
 3. ¿Qué opción de almacenamiento resulta más eficiente para las consultas a todos los cines? ¿Y sólo a Cine_cafés? En términos del número de operaciones de E/S, ¿cuánto más eficiente es la mejor de las técnicas para cada tipo de consulta en comparación con las otras dos técnicas?

Ejercicio 15.5 SGBDROO diferentes emplean técnicas diferentes para la creación de índices para la evaluación de consultas a jerarquías de colecciones. Para el ejemplo Pequeño hay dos opciones habituales para indexar los cines por su nombre:

- Crear un índice de árbol B+ para Cines.*nombre* y otro índice de árbol B+ para Cine_cafés.*nombre*.
 - Crear un índice de árbol B+ para la unión de Cines.*nombre* con Cine_cafés.*nombre*.
1. Describábase el modo de evaluar la consulta siguiente de manera eficiente empleando cada una de las opciones de indexación (esta consulta es para todos los tipos de tuplas de cines):

```
SELECT * FROM Cines C WHERE C.nombre = 'Majestic'
```

Dese una estimación del número de operaciones de E/S necesario en las dos situaciones posibles, suponiendo que hay 1 millón de cines normales y 1.000 cine-cafés. ¿Qué opción es más eficiente?

2. Hágase el mismo análisis para la consulta siguiente:

```
SELECT * FROM Cine_cafés C WHERE C.nombre = 'Majestic'
```

3. Para los índices agrupados, ¿interactúa la elección de técnica de indexación con la elección de opción de almacenamiento? ¿Y para los índices no agrupados?

Ejercicio 15.6 Considérese la consulta siguiente:

```
SELECT miniatura(I.imagen)
FROM Imágenes I
```

Dado que puede que la columna *I.imagen* contenga valores duplicados, describábase la manera de utilizar la asociación para evitar el cálculo de la función *miniatura* más de una vez para cada valor durante el procesamiento de esta consulta.

Ejercicio 15.7 Se dispone de un array bidimensional de objetos $n \times n$. Supóngase que se pueden encajar 100 objetos por página del disco. Describábase una manera de disponer (fragmentar) el array en las páginas de modo que las recuperaciones de subregiones cuadradas de $m \times m$ del array sean eficientes. (Las diferentes consultas solicitan subregiones de tamaños diferentes, es decir, valores diferentes de m , y la disposición del array en las páginas debe ofrecer un buen rendimiento, en promedio, para todas esas consultas.)

Ejercicio 15.8 Se le da a un optimizador de SGBDROO una consulta a una sola tabla con n condiciones de selección costosas, $\sigma_n(\dots(\sigma_1(T)))$. Para cada condición σ_i el optimizador puede estimar el coste c_i de su evaluación para una tupla y el factor de reducción de esa condición, r_i . Supóngase que hay t tuplas en T .

1. ¿Cuántas tuplas aparecen en el resultado de esta consulta?
2. Suponiendo que la consulta se evalúa tal y como puede verse (sin reordenar las condiciones de selección), ¿cuál es su coste total? Asegúrese de incluir el coste del examen de la tabla y el de la aplicación de las selecciones.
3. En el Apartado 15.8.2 se afirmó que el optimizador debe reordenar las operaciones de selección de modo que se apliquen a la tabla en orden de rangos crecientes, donde $\text{rango}_i = (r_i - 1)/c_i$. Demuéstrese que esta afirmación es óptima. Es decir, muéstrese que ninguna otra ordenación puede dar lugar a una consulta de coste inferior. (*Sugerencia*: puede que resulte más sencillo considerar primero el caso especial de que $n = 2$ y generalizar a partir de ahí.)

Ejercicio 15.9 Los SGBDROO soportan las referencias como tipo de datos. Se suele afirmar que el empleo de referencias en lugar de relaciones con claves y claves externas da lugar a un rendimiento mucho más elevado de las reuniones. Esta pregunta pretende que se explore ese problema.

- Considérese el siguiente DDL de SQL:1999 que sólo emplea estructuras relacionales puras:

```
CREATE TABLE R(claver integer, datosr text);
CREATE TABLE S(claves integer, claveer integer);
```

Supóngase que se tiene la siguiente consulta de reunión sencilla:

```
SELECT S.claveS, R.datosR
FROM   S, R
WHERE  S.claveER = R.claveR
```

- Considérese ahora el siguiente esquema de SGBDROO para SQL:1999:

```
CREATE TYPE t_r AS ROW(claveR integer, datosR text);
CREATE TABLE R OF t_r REF IS SYSTEM GENERATED;
CREATE TABLE S (claveS integer, R REF(t_r) SCOPE R);
```

Supóngase que se tiene la consulta siguiente:

```
SELECT S.claveS, S.R.claveR
FROM   S
```

¿Qué algoritmo sugeriría para evaluar la reunión de punteros del esquema del SGBDROO? ¿Cómo espera que se comporte en comparación con la reunión relacional sencilla del esquema anterior?

Ejercicio 15.10 Muchos sistemas relacionales orientados a objetos admiten los atributos evaluados como conjuntos mediante alguna variedad del constructor **setof**. Por ejemplo, suponiendo que se tiene el tipo **t_persona**, se puede crear la tabla Películas del esquema Pequeño de la Figura 15.1 de la manera siguiente:

```
CREATE TABLE Películas
  (nupelícula integer, título text, protagonistas setof Persona);
```

1. Describanse dos maneras de implementar los atributos evaluados como conjuntos. Una de ellas necesita registros de longitud variable, aunque todos los elementos del conjunto sean de longitud fija.
2. Examínense las consecuencias de las dos estrategias en la optimización de consultas con atributos evaluados como conjuntos.
3. Supóngase que se desea crear un índice para la columna **protagonistas** con objeto de buscar las películas por el nombre del artista que la ha protagonizado. Para ambas estrategias de implementación, analíicense estructuras de índices alternativas que puedan ayudar a acelerar esa consulta.
4. ¿Qué tipos de estadísticas debe mantener el optimizador de consultas para los atributos evaluados como conjuntos? ¿Cómo se pueden obtener?

NOTAS BIBLIOGRÁFICAS

Varias de las características orientadas a objetos que se han descrito aquí se basan en parte en ideas bastante antiguas de la comunidad de lenguajes de programación. [27] ofrece una buena visión general de estas ideas en el contexto de las bases de datos. El libro de Stonebraker [436] describe la visión de los SGBDROO incorporados en el producto pionero de su empresa, Illustra (ahora perteneciente a Informix). Entre los SGBD comerciales actuales con soporte relacional orientado a objetos están Universal Server de Informix, DB/2 CS V2 de IBM y UniSQL. Está previsto que una nueva versión de Oracle también incluya características para SGBDROO.

Muchas de las ideas de los sistemas relacionales orientados a objetos actuales surgieron de unos cuantos prototipos creados en los años 80 del siglo veinte, especialmente POSTGRES [440], Starburst [230] y O2 [149].

La idea de una base de datos orientada a objetos se formuló por primera vez en [129], que describía el sistema prototipo GemStone. Otros prototipos son DASDBS [393], EXODUS [78], IRIS [181], ObjectStore [291], ODE, [9] ORION [275], SHORE [77] y THOR [301]. O2 es realmente un ejemplo pionero de sistema que comenzaba a mezclar temas de los SGBDROO y de los SGBDOO —también podría incluirse en esta lista. [26] examina un conjunto de características que generalmente se considera que corresponden a los SGBDOO—. Entre los SGBDOO comercialmente disponibles en la actualidad figuran GemStone, Itasca, O2, Objectivity, ObjectStore, Ontos, Poet y Versant. [274] compara los SGBDOO con los SGBDR.

El soporte de las bases de datos a los TAD se exploró en primer lugar en los proyectos INGRES y POSTGRES de la U.C. Berkeley. Las ideas básicas se describen en [434], incluidos los mecanismos para el procesamiento de consultas y su optimización con TAD e indexación extensible. El soporte de los TAD se investigó también en el sistema de bases de datos Darmstadt [300]. El empleo correcto de la posibilidad de extensión de los índices de POSTGRES exigía un íntimo conocimiento de los mecanismos de las transacciones internas del SGBD. Se propusieron los árboles de búsqueda generalizados para resolver este problema; se describen en [249], mientras que los detalles de la concurrencia y de la recuperación basada en ARIES se presentaron en [285]. [402] propone que se debe permitir que los usuarios definan operadores para los objetos TAD y las propiedades de esos operadores que se pueden utilizar para la optimización de las consultas, en vez de un mero conjunto de métodos.

La fragmentación de los arrays se describe en [389]. [246, 104] presenta técnicas para el almacenamiento en memoria intermedia de métodos y para la optimización de consultas con métodos costosos. El almacenamiento de datos en la memoria intermedia en el lado del cliente de los SGBDOO cliente-servidor se estudia en [186]. La agrupación de objetos en el disco se estudia en [449]. El trabajo en las relaciones anidadas fue un lejano precursor de la investigación reciente en los objetos complejos de los SGBDOO y los SGBDROO. Una de las primeras proposiciones de relaciones anidadas es [311]. Los datos multivalorados desempeñan un papel importante en el razonamiento sobre la redundancia en las relaciones anidadas; véase, por ejemplo, [350]. Las estructuras de almacenamiento para las relaciones anidadas se estudiaron en [146].

Los modelos formales y los lenguajes de consulta para las bases de datos orientadas a objetos se han estudiado mucho; entre los trabajos que los tratan están [2, 38, 50, 74, 255, 256, 273, 349, 441]. [272] propone ampliaciones de SQL para la consulta de bases de datos orientadas a objetos. En GEM [482] se desarrolló una ampliación de SQL precursora y elegante con expresiones de ruta y herencia. Ha habido gran interés en la combinación de características deductivas y orientadas a objetos. Entre los trabajos en esta área figuran [28, 188, 305, 339, 428, 484]. Véase en el libro de texto [1] un análisis completo de los aspectos formales de la orientación a objetos y de los lenguajes de consultas.

[276, 277, 438, 487] incluye trabajos sobre SGBD que ahora se denominarían relacionales orientados a objetos o bien orientados a objetos. [485] contiene una visión general detallada de la evolución de los esquemas y de las bases de datos en los sistemas de bases de datos orientados a objetos. Se puede hallar una presentación completa de SQL:1999 en [327], mientras que las características avanzadas, incluidas las ampliaciones para objetos, se tratan en [325]. Se puede hallar un breve resumen de las nuevas características de SQL:1999 en [155]. La incorporación de varias características de SQL:1999 en DB2 de IBM se describe en [76]. OQL se describe en [87] y se basa en gran parte en el lenguaje de consultas O2, que se describe, junto con otros aspectos de O2, en el conjunto de trabajos [37].



16

ALMACENES DE DATOS Y AYUDA A LA TOMA DE DECISIONES

- ¿Por qué resultan inadecuados los SGBD tradicionales para la ayuda a la toma de decisiones?
- ¿Qué es el modelo de datos multidimensional y qué tipos de análisis facilita?
- ¿Qué características de SQL:1999 soportan las consultas multidimensionales?
- ¿Cómo soporta SQL:1999 el análisis de secuencias y de tendencias?
- ¿Cómo se optimizan los SGBD para que den respuestas rápidas para el análisis interactivo?
- ¿Qué tipos de organizaciones de índices y de archivos necesitan los sistemas OLAP?
- ¿Qué es el almacenamiento de datos y por qué es importante para la ayuda a la toma de decisiones?
- ¿Por qué se han vuelto importantes las vistas materializadas?
- ¿Cómo se pueden mantener de manera eficiente las vistas materializadas?
- **Conceptos fundamentales:** OLAP, modelo multidimensional, dimensiones, medidas; abstracción, concreción, pivotaje, tabulación cruzada, CUBE; consultas WINDOW, marcos, orden; consultas de los N primeros, agregación en línea; índices de mapas de bits, índices de reunión; almacenes de datos, extraer, actualizar, purgar; vistas materializadas, mantenimiento integral, mantenimiento de vistas en almacenes de datos.

Nada es más difícil y, por tanto, más precioso que ser capaz de decidir.

— Napoleón Bonaparte

Las organizaciones utilizan mucho los sistemas de gestión de bases de datos para conservar los datos que documentan sus operaciones cotidianas. En las operaciones que actualizan esos *datos operativos* las transacciones suelen realizar pequeñas modificaciones (por ejemplo,

añadir una reserva o ingresar un cheque) y se debe procesar gran número de transacciones de manera fiable y eficiente. Esas **aplicaciones de procesamiento de transacciones en línea** (online transaction processing, **OLTP**) han impulsado el crecimiento de la industria de los SGBD en las tres últimas décadas y, sin duda, seguirán teniendo importancia. Tradicionalmente, los SGBD se han optimizado ampliamente para tener un buen rendimiento en ese tipo de aplicaciones.

Recientemente, sin embargo, las organizaciones han ido poniendo el énfasis en las aplicaciones en las que se analizan y exploran conjuntamente datos actuales e históricos, se identifican tendencias útiles y se crean resúmenes de los datos con objeto de apoyar la toma de decisiones de alto nivel. Estas aplicaciones se denominan de **ayuda a la toma de decisiones**. Los principales fabricantes de SGBD relacionales han reconocido la importancia de este segmento del mercado y han añadido a sus productos características que lo soportan. En concreto, se ha ampliado SQL con nuevas estructuras y se han añadido técnicas novedosas de indexación y de optimización de consultas para dar soporte a las consultas complejas.

El empleo de vistas ha crecido rápidamente en popularidad debido a su utilidad en aplicaciones que implican el análisis de datos complejos. Aunque las consultas a las vistas se pueden responder evaluando la definición de la vista cuando se remite cada consulta, el cálculo previo de la definición de la vista puede hacer que las consultas se ejecuten mucho más rápido. Si la necesidad de las vistas precalculadas se lleva un paso más allá, las organizaciones pueden consolidar la información de varias bases de datos en un solo *almacén de datos* copiando las tablas de diversas procedencias en una sola ubicación o materializando vistas definidas sobre tablas de varios orígenes. El concepto de almacén de datos (*data warehouse*) se ha generalizado y ahora se dispone de muchos productos especializados para crear y administrar almacenes de datos desde varias bases de datos.

Este capítulo comienza con una visión general de la ayuda a la toma de decisiones en el Apartado 16.1. Se introducirá el modelo multidimensional de datos en el Apartado 16.2 y se tomarán en consideración los aspectos del diseño de bases de datos en el Apartado 16.2.1. Se estudiará la amplia clase de consultas que de forma natural soporta este modelo en el Apartado 16.3. Se analizará la manera en que las nuevas estructuras de SQL:1999 permiten expresar las consultas multidimensionales en el Apartado 16.3.1. En el Apartado 16.4 se estudian las extensiones de SQL:1999 que soportan las consultas a relaciones como conjuntos ordenados. Se considerará el modo de optimizar la generación rápida de las respuestas iniciales en el Apartado 16.5. Las muchas extensiones del lenguaje de consulta necesarias en el entorno OLAP impulsaron el desarrollo de nuevas técnicas de implementación; se analizarán en el Apartado 16.6. En el Apartado 16.7 se examinan los aspectos relativos a la creación y conservación de almacenes de datos. Desde un punto de vista técnico, un aspecto clave es la manera de conservar la información de los almacenes (tablas o vistas replicadas) cuando se modifica la fuente de información subyacente. Tras tratar el importante papel desempeñado por las vistas en OLAP y en el almacenamiento en el Apartado 16.8, se considerará el mantenimiento de las vistas materializadas en los apartados 16.9 y 16.10.

16.1 INTRODUCCIÓN A LA AYUDA A LA TOMA DE DECISIONES

La toma de decisiones en el marco de una organización exige una visión general de todos los aspectos de la empresa, por lo que muchas organizaciones han creado **almacenes de datos** consolidados que contienen datos obtenidos de diversas bases de datos de diferentes unidades de negocio junto con información histórica y resumida.

La tendencia hacia el almacenamiento de datos se complementa con un creciente énfasis en las herramientas de análisis potentes. Muchas características de las consultas de ayuda a la toma de decisiones hacen que los sistemas de SQL tradicionales resulten inadecuados:

- La cláusula WHERE suele contener muchas condiciones AND y OR. Las condiciones OR, en concreto, no son bien manejadas por muchos SGBD relacionales.
- Las aplicaciones exigen un amplio uso de las funciones estadísticas, como la desviación estándar, que no están soportadas en SQL-92. Por tanto, las consultas SQL deben incrustarse a menudo en programas de un lenguaje anfitrión.
- Muchas consultas implican condiciones sobre el tiempo o exigen la agregación a lo largo de períodos de tiempo. SQL-92 ofrece poco soporte para este análisis de series temporales.
- Los usuarios suelen necesitar plantear varias consultas relacionadas. Dado que no hay manera adecuada de expresar estas familias de consultas que aparecen a menudo, los usuarios tienen que escribirlas como conjuntos de consultas independientes, lo que puede resultar tedioso. Además, el SGBD no tiene modo alguno de reconocer y aprovechar las oportunidades de organización que surgen de la ejecución conjunta de muchas consultas relacionadas.

Se dispone de tres amplias clases de herramientas de análisis. En primer lugar, algunos sistemas soportan una clase de consultas estilizadas que suelen implicar operadores de agrupación y de agregación y ofrecen un soporte excelente para las condiciones booleanas complejas, funciones estadísticas y características para el análisis de series temporales. Las aplicaciones dominadas por estas consultas se denominan **procesamiento analítico en línea** (online analytic processing, **OLAP**). Estos sistemas soportan un estilo de consulta en el que resulta más conveniente pensar en los datos como en arrays multidimensionales y están influidos por herramientas para los usuarios finales como las hojas de cálculo, además de por los lenguajes de consulta de las bases de datos.

En segundo lugar, algunos SGBD soportan consultas tradicionales al estilo SQL pero están diseñados para soportar también consultas OLAP de manera eficiente. Estos sistemas se pueden considerar como SGBD relacionales optimizados para las aplicaciones de ayuda a la toma de decisiones. Muchos fabricantes de SGBD están mejorando actualmente sus productos en este sentido y, con el tiempo, es probable que disminuya la diferencia entre los sistemas OLAP especializados y los SGBD relacionales mejorados para soportar consultas OLAP.

La tercera clase de herramientas de análisis está motivada por el deseo de hallar tendencias y pautas interesantes o inesperadas en conjuntos de datos de gran tamaño en lugar de las complejas características de las consultas que se acaban de referir. En el **análisis explorador de datos**, aunque los analistas puedan reconocer las “pautas interesantes” cuando

SQL:1999 y OLAP. En este capítulo se analizarán varias características introducidas en SQL:1999 para soportar OLAP. A fin de no retrasar la publicación de la norma de SQL:1999, estas características se añadieron realmente a la norma mediante una *enmienda* denominada SQL/OLAP.

se les muestran, resulta muy difícil formular consultas que capturen la esencia de las pautas interesantes. Por ejemplo, puede que un analista que examine historiales de uso de tarjetas de crédito desee detectar una actividad inusual que indique el mal empleo de una tarjeta perdida o robada. Puede que un vendedor por catálogo desee examinar los registros de los clientes para identificar clientes prometedores para una nueva promoción; esta identificación dependerá del nivel de renta, de las pautas de compra, de las áreas por las que haya mostrado interés, etcétera. La cantidad de datos en muchas aplicaciones es demasiado grande para permitir el análisis manual o, ni siquiera, el análisis estadístico tradicional, y el objetivo de la **minería de datos** es apoyar el análisis explorador en conjuntos de datos de tamaño muy grande. La minería de datos se estudia más a fondo en el Capítulo 17.

Evidentemente, es probable que la evaluación de consultas OLAP o de minería de datos en datos distribuidos globalmente sea exasperantemente lenta. Además, para análisis tan complejos, que suelen ser de naturaleza estadística, no resulta fundamental que se utilice la versión más reciente de los datos. La solución natural es crear un repositorio central de todos los datos; es decir, un almacén de datos. Así, la disponibilidad del almacén facilita la aplicación de las herramientas OLAP y de minería de datos y, a la inversa, el deseo de aplicar esas herramientas de análisis es un motivo de peso para la creación de almacenes de datos.

16.2 OLAP: MODELO MULTIDIMENSIONAL DE DATOS

Las aplicaciones OLAP están dominadas por consultas complejas *ad hoc*. En términos de SQL, se trata de consultas que implican operadores de agrupación y de agregación. La manera natural de considerar las consultas OLAP típicas, no obstante, es hacerlo en términos de un modelo multidimensional de datos. En este apartado se presentará el modelo multidimensional de datos y se comparará con una representación relacional de los datos. En apartados posteriores se describirán las consultas OLAP en términos del modelo multidimensional de datos y se considerarán algunas técnicas nuevas de implementación diseñadas para soportar esas consultas.

En el modelo multidimensional de datos la atención se centra en un conjunto de **medidas** numéricas. Cada medida depende de un conjunto de **dimensiones**. Se desarrollará un ejemplo basado en datos de ventas. El atributo de medida del ejemplo es *ventas*. Las dimensiones son Productos, Ubicaciones y Horas. Dado un producto, una ubicación y una hora se tiene, como máximo, un valor de ventas asociado. Si se identifica cada producto mediante el identificador único *idp* y, análogamente, se identifica la ubicación mediante *idubi* y la hora mediante *idhora*, se puede pensar en la información de ventas como si estuviera dispuesta en el array tridimensional Ventas. Este array puede verse en la Figura 16.1; en aras de la

claridad sólo se muestran los valores para un único valor de *idubi*, *idubi*= 1, que se puede considerar un corte ortogonal al eje *idubi*.

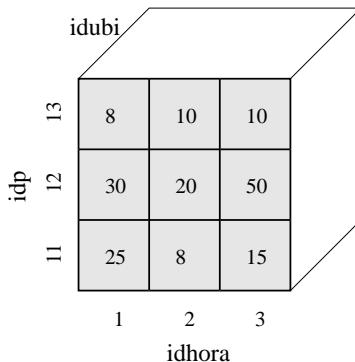


Figura 16.1 Ventas: un conjunto de datos multidimensional

Esta visión de los datos como un array multidimensional se puede generalizar fácilmente a más de tres dimensiones. En las aplicaciones OLAP la mayor parte de los datos se puede representar en un array multidimensional de este tipo. En realidad, algunos sistemas OLAP guardan realmente datos en un array multidimensional (por supuesto, implementado sin la suposición habitual de los lenguajes de programación de que todo el array cabe en la memoria). Los sistemas OLAP que utilizan arrays para guardar los conjuntos de datos multidimensionales se denominan sistemas **OLAP multidimensionales** (multidimensional OLAP, **MOLAP**).

Los datos de un array multidimensional también se pueden representar en forma de relación, como se ilustra en la Figura 16.2, que muestra los mismos datos que la Figura 16.1, con filas adicionales correspondientes al “corte” *idubi*= 2. Esta relación, que relaciona las dimensiones con la medida de interés, se denomina **tabla de hechos**.

Considérense ahora las dimensiones. Cada dimensión puede tener un conjunto de atributos asociados. Por ejemplo, la dimensión Ubicaciones se identifica mediante el atributo *idubi*, que se utilizó en la tabla Ventas para identificar ubicaciones. Se da por supuesto que también tiene los atributos *país*, *provincia* y *ciudad*. Además, se supone que la dimensión Productos tiene los atributos *nombrep*, *categoría* y *precio* además del identificador *idp*. La *categoría* de cada producto indica su naturaleza general; por ejemplo, el producto *pantalones* podría tener el valor de categoría *ropa*. Se da por supuesto que la dimensión Horas tiene los atributos *fecha*, *semana*, *mes*, *trimestre*, *año* y *indicador_festivo*, además del identificador *idhora*.

Para cada dimensión el conjunto de valores asociados se puede estructurar de manera jerárquica. Por ejemplo, las ciudades pertenecen a provincias y las provincias pertenecen a países. Las fechas pertenecen a semanas y a meses, tanto semanas como meses están contenidas en trimestres y los trimestres están contenidos en años. (Téngase en cuenta que una semana puede hallarse a caballo de dos meses; por tanto, las semanas no están contenidas en los meses.) Algunos de los atributos de las dimensiones describen la posición del valor de la dimensión con respecto a esta jerarquía subyacente de valores de las dimensiones. Las

<i>idubi</i>	<i>ciudad</i>	<i>provincia</i>	<i>país</i>
1	Madison	C	España
2	Fresno	H	España
5	Chennai	TN	India

Ubicaciones

<i>idp</i>	<i>nombrep</i>	<i>categoría</i>	<i>precio</i>
11	Jeans Lee	Ropa	25
12	Zord	Juguetes	18
13	Biro Pen	Papelería	2

Productos

<i>idp</i>	<i>idhora</i>	<i>idubi</i>	<i>ventas</i>
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
13	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35
11	2	2	22
11	3	2	10
12	1	2	26
12	2	2	45
12	3	2	20
13	1	2	20
13	2	2	40
13	3	2	5

Ventas

Figura 16.2 Ubicaciones, Productos y Ventas representados como relaciones

jerarquías para las dimensiones Productos, Ubicaciones y Horas del ejemplo se muestran en el nivel de atributos de la Figura 16.3.

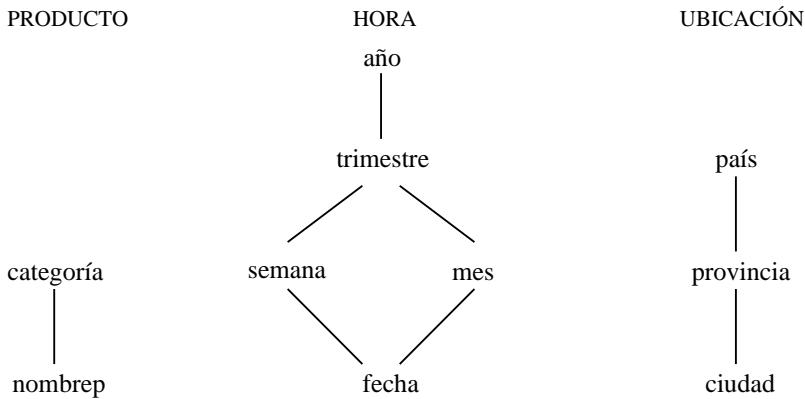


Figura 16.3 Jerarquías de las dimensiones

La información sobre las dimensiones también se puede representar en forma de conjunto de relaciones:

```

    Ubicaciones(idubi: integer, ciudad: string, provincia: string, país: string)
    Productos(idp: integer, nombrep: string, categoría: string, precio: real)
    Horas(idhora: integer, fecha: string, semana: integer, mes: integer,
          trimestre: integer, año: integer, indicador_festivo: boolean )
  
```

Estas relaciones son mucho más pequeñas que la tabla de hechos de cualquier aplicación OLAP típica; se denominan **tablas de dimensiones**. Los sistemas OLAP que guardan toda la información, incluidas las tablas de hechos, en forma de relación se denominan sistemas **OLAP relacionales** (relational OLAP, ROLAP).

La tabla Horas ilustra la atención prestada a la dimensión Horas en las aplicaciones OLAP típicas. Los tipos de datos de SQL date y timestamp no resultan adecuados; para soportar los resúmenes que reflejan las operaciones comerciales se conserva información como los trimestres fiscales, la condición de festivo, etcétera para cada valor de hora.

16.2.1 Diseño de bases de datos multidimensionales

La Figura 16.4 muestra las tablas del ejemplo de ventas. Sugiere una estrella, centrada en la tabla de hechos Ventas; esta combinación de tabla de hechos y de tabla de dimensiones se denomina **esquema en estrella**. Esta pauta de esquema es muy frecuente en las bases de datos diseñadas para OLAP. La mayor parte de los datos suele hallarse en la tabla de hechos, que no tiene redundancia; suele estar en la FNBC. De hecho, para minimizar el tamaño de la tabla de hechos, los identificadores de las dimensiones (como *idp* e *idhora*) son identificadores generados por el sistema.

La información sobre el valor de las dimensiones se conserva en las tablas de dimensiones. Las tablas de dimensiones no suelen estar normalizadas. La explicación es que las tablas de

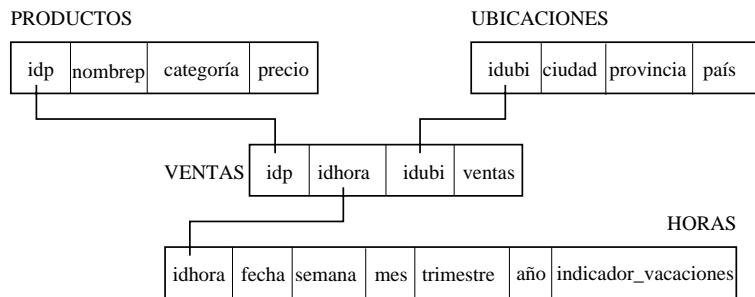


Figura 16.4 Ejemplo de esquema en estrella

dimensiones de las bases de datos empleadas para OLAP son estáticas y que las anomalías de actualización, de inserción o de borrado no son importantes. Además, como el tamaño de la base de datos está determinado sobre todo por la tabla de hechos, el espacio que se ahorra normalizando las tablas de dimensiones es despreciable. Por tanto, el principal criterio de diseño es minimizar el tiempo de cálculo para la combinación de los hechos de la tabla de hechos con la información sobre las dimensiones, lo que sugiere que se evita descomponer las tablas de dimensiones en otras más pequeñas (lo que podría llevar a reuniones adicionales).

Los tiempos de respuesta cortos para las consultas interactivas son importantes en OLAP, y la mayor parte de los sistemas soportan la materialización de las tablas resumen (que suelen generarse mediante consultas que emplean el agrupamiento). Las consultas *ad hoc* formuladas por los usuarios se responden empleando las tablas originales junto con los resúmenes calculados previamente. Un problema muy importante del diseño es seleccionar las tablas que se deben materializar para conseguir un empleo óptimo de la memoria disponible y contestar las consultas *ad hoc* formuladas frecuentemente con tiempos de respuesta interactivos. En los sistemas OLAP actuales puede que la elección de las tablas resumen que se deben materializar sea la decisión de diseño más importante.

Finalmente, se han desarrollado nuevas estructuras de almacenamiento y técnicas de indexación para soportar OLAP que ofrecen al diseñador de bases de datos más opciones de diseño físico. Algunas de estas técnicas de implementación se tratan en el Apartado 16.6.

16.3 CONSULTAS DE AGREGACIÓN MULTIDIMENSIONALES

Ahora que se ha examinado el modelo de datos multidimensional, se considerará el modo en que se pueden consultar y manipular esos datos. Las operaciones soportadas por este modelo están muy influidas por las herramientas de los usuarios finales, como las hojas de cálculo. El objetivo es ofrecer a los usuarios finales que no son expertos en SQL una interfaz intuitiva y potente para las tareas de análisis habituales orientadas al mundo empresarial. Se espera que los usuarios formulen directamente consultas *ad hoc*, sin depender de los programadores de aplicaciones de bases de datos.

En este apartado se da por supuesto que el usuario trabaja con un conjunto de datos multidimensional y que cada operación devuelve una presentación diferente o un resumen;

el conjunto de datos subyacente está siempre disponible para que el usuario lo manipule, independientemente del nivel de detalle con el que se esté examinando. En el Apartado 16.3.1 se estudia el modo en que SQL:1999 ofrece estructuras para expresar los tipos de consultas que se presentan en este apartado sobre datos relacionales tabulares.

Una operación muy frecuente es la agregación de medidas a una o varias dimensiones. Las consultas siguientes son típicas:

- Averiguar las ventas totales.
- Averiguar las ventas totales de cada ciudad.
- Averiguar las ventas totales de cada provincia.

Estas consultas se pueden expresar como consultas SQL sobre las tablas de hechos y de dimensiones. Cuando se agregan medidas en una o varias dimensiones, la medida agregada depende de menos dimensiones que las originales. Por ejemplo, cuando se calculan las ventas totales por ciudad, la medida agregada es *ventas totales* y sólo depende de la dimensión Ubicaciones, mientras que la medida original, *ventas*, dependería de las dimensiones Ubicaciones, Horas y Productos.

Otro empleo de la agregación es la obtención de resúmenes en diferentes niveles de una jerarquía de dimensiones. Si se tienen las ventas totales por ciudad, se pueden agregar sobre la dimensión Ubicaciones para obtener las ventas por provincia. Esta operación se denomina **abstracción** en la literatura OLAP. Lo contrario del enrollado es la **concreción**: dadas las ventas totales por estado, se puede pedir una representación más detallada si se concreta por Ubicaciones. Se pueden pedir las ventas por cada ciudad o sólo las ventas por ciudad para una provincia determinada (con las ventas presentadas provincia por provincia para las demás ciudades, como antes). También se pueden concretar otras dimensiones distintas de Ubicaciones. Por ejemplo, se pueden pedir las ventas totales de cada producto en cada provincia, concretando la dimensión Productos.

Otra operación frecuente es el **pivotaje**. Considérese una representación tabular de la tabla Ventas. Si se crean tablas dinámicas para las dimensiones Ubicaciones y Horas se obtiene una tabla de ventas totales para cada ubicación y cada valor de hora. Esta información se puede representar en forma de gráfico bidimensional en el que los ejes se etiquetan con los valores de ubicación y de hora; los valores del gráfico corresponden a las ventas totales para esa ubicación y momento. Por tanto, los valores que aparecen en las columnas de la representación original se transforman en etiquetas de los ejes en la presentación resultante. El resultado de la creación de tablas dinámicas, denominado **tabla de doble entrada**, se muestra en la Figura 16.5. Obsérvese que en la presentación como hoja de cálculo, además de las ventas totales por año y por provincia (tomadas en conjunto), también hay otros resúmenes de ventas por año y por provincia.

El pivotaje también se puede utilizar para modificar las dimensiones de las tablas de doble entrada; a partir de la presentación de las ventas por año y por provincia se puede obtener la presentación de las ventas por producto y por año.

Evidentemente, el entramado OLAP facilita la presentación de una amplia clase de consultas. También da nombres atractivos a algunas operaciones conocidas: **Cortar** un conjunto de datos equivale a realizar una selección de igualdad en una o varias dimensiones, posiblemente junto con la proyección externa de algunas dimensiones. **Cortar en cubos** un conjunto de

	C	H	Total
1995	63	81	144
1996	38	107	145
1997	75	35	110
Total	176	223	399

Figura 16.5 Tabla de doble entrada de ventas por año y por provincia

datos equivale a realizar una selección de rangos. Estos términos provienen de la visualización del efecto de estas operaciones en la representación de los datos en cubos o tablas de doble entrada.

Una nota sobre las bases de datos estadísticas

Muchos conceptos OLAP se hallan presentes en trabajos anteriores sobre las **bases de datos estadísticas** (BDE), que son sistemas de bases de datos diseñados para soportar aplicaciones estadísticas, aunque esta conexión no haya sido suficientemente reconocida debido a las diferencias en los dominios de aplicación y en la terminología. El modelo multidimensional de datos, con los conceptos de medida asociada a las dimensiones y de jerarquías de clasificación para los valores de las dimensiones, también se emplea en las BDE. Las operaciones OLAP como la abstracción y la concreción tienen sus equivalentes en las BDE. En realidad, parte de las técnicas de implementación desarrolladas para OLAP se aplica también a las BDE.

No obstante, surgen algunas diferencias debido a los diferentes dominios para cuyo soporte se desarrollaron OLAP y las BDE. Por ejemplo, las BDE se utilizan en aplicaciones de socioeconomía, en las que las jerarquías de clasificación y los problemas de intimidad son muy importantes. Esto se refleja en la mayor complejidad de las jerarquías de clasificación de las BDE, junto con problemas como las posibles violaciones de la intimidad. (El problema de la intimidad afecta al problema de si un usuario con acceso a datos resumidos puede reconstruir los datos originales sin resumir.) Por el contrario, OLAP se ha dirigido a aplicaciones empresariales con grandes volúmenes de datos, y el tratamiento eficiente de conjuntos de datos de tamaño muy grande ha recibido más atención que en la literatura sobre las BDE.

16.3.1 ROLLUP y CUBE en SQL:1999

En este apartado se analizará cuántas de las posibilidades de consulta del modelo multidimensional están soportadas en SQL:1999. Generalmente una sola operación OLAP provoca varias consultas SQL estrechamente relacionadas con la agregación y la agrupación. Por ejemplo, considérese la tabla de doble entrada de la Figura 16.5, que se obtuvo mediante pivotaje basado en la tabla Ventas. Para obtener la misma información, habría que formular las consultas siguientes:

```
SELECT H.año, U.provincia, SUM (V.ventas)
```

```

FROM      Ventas V, Horas H, Ubicaciones U
WHERE     V.idhora=H.idhora AND V.idubi=U.idubi
GROUP BY  H.añoo, U.provincia

```

Esta consulta genera las entradas en la parte principal del gráfico (esbozado mediante las líneas oscuras). La columna resumen de la derecha se genera mediante la consulta:

```

SELECT    H.año, SUM (V.ventas)
FROM      Ventas V, Horas H
WHERE     v.idhora=H.idhora
GROUP BY  H.año

```

La fila resumen de la parte inferior se genera mediante la consulta:

```

SELECT    U.provincia, SUM (V.ventas)
FROM      Ventas V, Ubicaciones U
WHERE     V.idubi=U.idubi
GROUP BY  U.provincia

```

La suma acumulada de la esquina inferior derecha del gráfico se produce mediante la consulta:

```

SELECT    SUM (V.ventas)
FROM      Ventas V, Ubicaciones U
WHERE     V.idubi=U.idubi

```

La tabla de doble entrada de ejemplo se puede considerar como una abstracción de todo el conjunto de datos (es decir, que trata todo como un gran grupo), sobre la dimensión Ubicaciones, sobre la dimensión Horas y sobre las dimensiones Ubicaciones y Horas conjuntamente. Cada abstracción corresponde a una sola consulta SQL con agrupación. En general, dada una medida con k dimensiones asociadas, se puede abstraer sobre un subconjunto de cualquiera de esas k dimensiones; por tanto, se tiene un total de 2^k consultas SQL de este tipo.

Mediante operaciones de alto nivel, como el pivotaje, los usuarios pueden generar muchas de estas 2^k consultas SQL. El reconocimiento de las coincidencias entre estas consultas permite un cálculo más eficiente y coordinado del conjunto de consultas.

SQL:1999 amplía la estructura GROUP BY para que ofrezca mejor soporte de la abstracción y de las consultas de tablas de doble entrada. La cláusula GROUP BY con la palabra clave CUBE es equivalente a un conjunto de instrucciones GROUP BY, con una instrucción GROUP BY por cada subconjunto de las k dimensiones.

Considérese la consulta siguiente:

```

SELECT    H.año, U.provincia, SUM (V.ventas)
FROM      Ventas V, Horas H, Ubicaciones U
WHERE     V.idhora=H.idhora AND V.idubi=U.idubi
GROUP BY  CUBE (H.año, U.provincia)

```

El resultado de esta consulta, que puede verse en la Figura 16.6, es precisamente una representación tabular de las tablas de doble entrada de la Figura 16.5.

<i>H.año</i>	<i>U.provincia</i>	$SUM(V.ventas)$
1995	C	63
1995	H	81
1995	<i>null</i>	144
1996	C	38
1996	H	107
1996	<i>null</i>	145
1997	C	75
1997	H	35
1997	<i>null</i>	110
<i>null</i>	C	176
<i>null</i>	H	223
<i>null</i>	<i>null</i>	399

Figura 16.6 Resultado de GROUP BY CUBE para Ventas

SQL:1999 ofrece también variedades de GROUP BY que permiten el cálculo de subconjuntos de la tabla de doble entrada calculada mediante GROUP BY CUBE. Por ejemplo, se puede sustituir la cláusula de agrupamiento de la consulta anterior por

GROUP BY ROLLUP (H.año, U.provincia)

A diferencia de GROUP BY CUBE, se agrega según todas las parejas de valores de año y provincia y según cada año, y se calcula una suma global para todo el conjunto de datos (la última fila de la Figura 16.6), pero no se agrega para cada valor de provincia. El resultado es idéntico al mostrado en la Figura 16.6, salvo que las filas con *null* en la columna *H.año* y valores no *null* en la columna *U.provincia* no se calculan.

CUBE idp, idubi, idhora BY SUM Ventas

Esta consulta concreta la tabla Ventas para los ocho subconjuntos del conjunto {idp, idubi, idhora} (incluido el conjunto vacío). Es equivalente a ocho consultas de la forma

```
SELECT    SUM (V.ventas)
FROM      Ventas V
GROUP BY  lista de agrupación
```

Las consultas sólo se diferencian en la *lista de agrupación*, que es un subconjunto del conjunto {idp, idubi, idhora}. Se puede considerar que estas ocho consultas están distribuidas por un retículo, como puede verse en la Figura 16.7. Las tuplas resultado de cada nodo se pueden seguir agregando para calcular el resultado para cualquier vástago de ese nodo. Esta relación entre las consultas que surgen de una instrucción CUBE se puede aprovechar para obtener una evaluación eficiente.

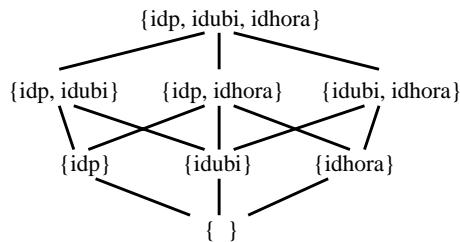


Figura 16.7 El retículo de las consultas GROUP BY en una consulta CUBE

16.4 CONSULTAS VENTANA EN SQL:1999

La dimensión temporal es muy importante en la ayuda a la toma de decisiones, y las consultas que implican análisis de tendencias han sido tradicionalmente difíciles de expresar en SQL. Para abordar esto, SQL:1999 introdujo una ampliación fundamental denominada **consulta ventana**. Entre los ejemplos de consultas que se pueden escribir empleando esta ampliación pero que son difíciles o imposibles de escribir sin ella en SQL figuran

1. Averiguar las ventas totales por meses.
2. Averiguar las ventas totales por meses para cada ciudad.
3. Averiguar la variación porcentual de las ventas mensuales totales para cada producto.
4. Averiguar los cinco productos principales, ordenados por ventas totales.
5. Averiguar la media móvil retrasada de n días de las ventas. (Para cada día hay que calcular las ventas diarias promedio a lo largo de los n días anteriores.)
6. Averiguar los cinco productos principales ordenados por ventas acumuladas para cada mes del año pasado.
7. Ordenar todos los productos por ventas totales a lo largo del año pasado y, para cada producto, imprimir la diferencia en ventas totales con respecto al producto que lo sigue en la clasificación.

Las dos primeras consultas se pueden expresar como consultas SQL empleando GROUP BY con las tablas de hechos y de dimensiones. Las dos consultas siguientes también se pueden expresar en SQL-92, pero resultan bastante complicadas. La quinta consulta no se puede expresar en SQL-92 si n tiene que ser uno de los parámetros de la consulta. La última consulta no se puede expresar en SQL-92.

En este apartado se estudian las características de SQL:1999 que permiten expresar todas estas consultas y, evidentemente, una amplia gama de consultas similares.

La ampliación principal es la cláusula WINDOW, que identifica de manera intuitiva una “ventana” ordenada de filas “alrededor” de cada tupla de una tabla. Esto permite aplicar una amplia gama de funciones agregadas a la ventana de una fila y ampliar la fila con esos resultados. Por ejemplo, se pueden asociar las ventas promedio de los últimos 3 días con todas

las tuplas de Ventas (cada una de las cuales registra las ventas de 1 día). Esto proporciona una media móvil de las ventas a lo largo de tres días.

Aunque hay cierta similitud con las cláusulas GROUP BY y CUBE, también existen importantes diferencias. Por ejemplo, al igual que el operador WINDOW, GROUP BY permite crear particiones de filas y aplicar funciones de agregación como SUM a las filas de las particiones. Sin embargo, a diferencia de WINDOW, hay una sola fila de resultados por partición, en vez de una por fila, y cada partición es un conjunto desordenado de filas.

Ahora se ilustrará el concepto de ventana mediante un ejemplo:

```
SELECT U.provincia, H.mes, AVG (V.ventas) OVER W AS mediamóv
FROM   Ventas V, Horas H, Ubicaciones U
WHERE  V.idhora=H.idhora AND V.idubi=U.idubi
WINDOW W AS (PARTITION BY U.provincia
              ORDER BY H.mes
              RANGE BETWEEN INTERVAL '1' MONTH PRECEDING
              AND INTERVAL '1' MONTH FOLLOWING)
```

Las cláusulas FROM y WHERE se procesan como siempre para generar (conceptualmente) una tabla intermedia, a la que nos referiremos como Temp. Se crean ventanas alrededor de la relación Temp.

Hay tres etapas en la definición de una ventana. En primer lugar se definen las *particiones* de la tabla, mediante la cláusula PARTITION BY. En el ejemplo, las particiones se basan en la columna *U.provincia*. Las particiones son similares a los grupos creados con GROUP BY, pero hay una diferencia muy importante en el modo en que se procesan. Para comprender la diferencia, obsérvese que la cláusula SELECT contiene una columna, *H.mes*, que no se emplea para definir las particiones; filas diferentes de una misma partición podrían tener valores diferentes en esta columna. Una columna así no puede aparecer en la cláusula SELECT junto con la agrupación, pero se permite que lo haga para las particiones. El motivo es que hay una fila de resultados por *cada* fila de la partición de Temp, en vez de sólo una fila de resultados por partición. La ventana centrada en cada fila dada se utiliza para calcular las funciones de agregación de la fila de resultados correspondiente.

La segunda etapa de la definición de una ventana es especificar el *orden* de las filas en la cada partición. Esto se lleva a cabo mediante la cláusula ORDER BY; en nuestro el, las filas de cada partición se ordenan según *H.mes*.

La tercera etapa de la definición de una ventana es el *enmarcado* de las ventanas; es decir, establecer los límites de la ventana asociada con cada fila en términos de la ordenación de las filas dentro de las particiones. En el ejemplo la ventana de cada fila incluye a la propia fila y a todas las filas cuyo valor de mes se halle entre el inmediatamente anterior y el inmediatamente posterior al de esa fila; por tanto, una fila cuyo valor de mes sea junio de 2002 tendrá una ventana que contenga todas las filas con mes igual a mayo, junio o julio de 2002.

La fila de resultados correspondiente a una fila dada se crea identificando primero su ventana. Luego, para cada columna de resultados definida mediante una función de agregación de ventana, se calcula el agregado correspondiente empleando las filas de esa ventana.

En este ejemplo cada fila de Temp es básicamente una fila de Ventas, etiquetada con detalles adicionales (sobre las dimensiones ubicación y hora). Hay una partición para cada provincia y cada fila de Temp pertenece exactamente a una partición. Considérese una fila

para una tienda de Coruña. La fila indica las ventas de un producto determinado en esa tienda en un momento determinado. La ventana de esa fila incluye todas las filas que describen las ventas en Coruña entre el mes previo y el siguiente y *mediamóv* es la media de ventas (para todos los productos) en Coruña en ese periodo.

Hay que destacar que el orden de las filas en cada partición con objeto de definir la ventana no se extiende a la tabla de filas de resultados. El orden de las filas de resultados no es determinista, a menos, por supuesto, que se capturen mediante un cursor y se emplee `ORDER BY` para ordenar el resultado de ese cursor.

16.4.1 Enmarcado de ventanas

Existen dos maneras diferentes de enmarcar ventanas en SQL:1999. La consulta de ejemplo ilustraba la estructura `RANGE`, que define una ventana basada en el valor de alguna columna (*mes* en el ejemplo). La columna que establece el orden tiene que ser de tipo numérico, fecha-hora o intervalo, ya que se trata de los únicos tipos para los que están definidas la suma y la resta.

El segundo enfoque se basa en el empleo directo de la ordenación y en especificar el número de filas anteriores y posteriores a la dada se hallan en su ventana. Por tanto, se puede decir

```
SELECT U.provincia, H.mes, AVG (V.ventas) OVER W AS mediamóv
  FROM Ventas V, Horas H, Ubicaciones U
 WHERE V.idhora=H.idhora AND V.idubi=U.idubi
WINDOW W AS (PARTITION BY U.provincia
              ORDER BY H.mes
              ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
```

Si hay exactamente una fila en Temp para cada mes, esto es equivalente a la consulta anterior. Sin embargo, si un mes dado no tiene filas o tiene varias, las dos consultas dan resultados diferentes. En este caso, el resultado de la segunda consulta es difícil de comprender porque las ventanas para las diferentes filas no se alinean de manera natural.

El segundo enfoque resulta adecuado si, en términos de este ejemplo, hay exactamente una fila por mes. Generalizando a partir de esto, también resulta apropiado si hay exactamente una fila por cada valor de la secuencia de valores de la columna que establece el orden. A diferencia del primer enfoque, en el que el orden debe especificarse en una sola columna (de tipo numérico, fecha-hora o intervalo) el orden se puede basar en claves compuestas.

También se pueden definir ventanas que incluyan todas las filas anteriores a una fila dada (`UNBOUNDED PRECEDING`) o posteriores a ella (`UNBOUNDED FOLLOWING`) dentro de la partición de cada fila.

16.4.2 Nuevas funciones de agregación

Aunque las funciones de agregación estándar que se aplican a los multiconjuntos de valores (por ejemplo, `SUM`, `AVG`) se pueden utilizar junto con la creación de ventanas, hace falta una nueva clase de funciones que opere con *listas* de valores.

La función `RANK` devuelve la posición de cada fila dentro de su partición. Si una partición tiene 15 filas, la primera (según el orden de las filas de la definición de ventana para esta

partición) tiene rango 1 y la última tiene rango 15. El rango de las filas intermedias depende de si hay filas múltiples para algún valor de la columna que establece el orden.

Considérese de nuevo el ejemplo anterior. Si la primera fila de la partición Coruña tiene el mes de enero de 2002 y tanto la segunda como la tercera filas tienen el mes de febrero de 2002, sus rangos son 1, 2 y 2, respectivamente. Si la siguiente fila tiene marzo de 2002, su rango es 4.

Por el contrario, la función `DENSE_RANK` genera rangos sin saltos. En el ejemplo, las cuatro filas reciben los rangos 1, 2, 2 y 3. La única modificación se halla en la cuarta fila, cuyo rango es ahora 3 en vez de 4.

La función `PERCENT_RANK` da una medida de la posición relativa de cada fila en la partición. Se define como $(\text{RANK}-1)$ dividido por el número de filas de la partición. `CUME_DIST` es parecida, pero se basa en la posición real dentro de la partición ordenada en vez de hacerlo en el rango.

16.5 BÚSQUEDA RÁPIDA DE RESPUESTAS

Una tendencia reciente, alimentada en parte por la popularidad de Internet, es el énfasis en las consultas de las que el usuario sólo desea obtener rápidamente las primeras, o las “mejores”, respuestas. Cuando los usuarios formulan consultas a motores de búsqueda como AltaVista, rara vez examinan más allá de la primera o segunda página de resultados. Si no hallan lo que buscan, refinan la consulta y la vuelven a enviar. Se produce el mismo fenómeno en las aplicaciones de apoyo a la toma de decisiones, y algunos productos de SGBD (por ejemplo, DB2) ya soportan estructuras de SQL ampliadas para que se especifiquen esas consultas. Una tendencia relacionada es que, para las consultas complejas, a los usuarios les gustaría ver rápidamente una respuesta aproximada y luego poder refinarla continuamente, en vez de esperar hasta que esté disponible la respuesta exacta. Estas dos tendencias se examinarán brevemente a continuación.

16.5.1 Las consultas de los N primeros

Los analistas suelen querer identificar el puñado de productos más vendidos, por ejemplo. Se puede ordenar por ventas para cada producto y devolver las respuestas en ese orden. Si tenemos un millón de productos y el analista sólo está interesado en los 10 primeros, esta estrategia directa de evaluación es, claramente, un derroche. Es deseable que los usuarios puedan indicar de manera explícita el número de respuestas que desean, lo que hará posible que el SGBD optimice la ejecución. La siguiente consulta de ejemplo pide los 10 primeros productos de una ubicación y una hora dadas ordenados por ventas:

```
SELECT P.idp, P.nombrep, V.ventas
  FROM Ventas V, Productos P
 WHERE V.idp=P.idp AND V.idubi=1 AND V.idhora=3
 ORDER BY V.ventas DESC
OPTIMIZE FOR 10 ROWS
```

La estructura `OPTIMIZE FOR N ROWS` no se halla en SQL-92 (ni siquiera en SQL:1999), pero está soportado en el producto DB2 de IBM, y otros productos (por ejemplo, Oracle 9i) tienen estructuras similares. En ausencia de pistas como `OPTIMIZE FOR 10 ROWS`, el SGBD

calcula las ventas para todos los productos y las devuelve en orden descendente de ventas. La aplicación puede cerrar el cursor de resultados (es decir, terminar la ejecución de la consulta) tras consumir 10 filas, pero ya se habrá realizado un esfuerzo considerable en calcular las ventas para todos los productos y ordenarlas.

Considérese ahora el modo en que un SGBD puede utilizar la sugerencia `OPTIMIZE FOR` para ejecutar la consulta de manera eficiente. La clave está en calcular de algún modo sólo las ventas para los productos que tengan probabilidades de estar entre los 10 primeros en ventas. Supóngase que se conoce la distribución de los valores de ventas porque se conserva un histograma para la columna *ventas* de la relación Ventas. Entonces, se puede escoger un valor de *ventas*, por ejemplo, c , tal que sólo 10 productos tengan un valor de ventas más elevado. A las tuplas de Ventas que cumplan esa condición se les pueden aplicar también las condiciones de ubicación y de hora y ordenar el resultado. La evaluación de la consulta siguiente resulta equivalente a este enfoque:

```
SELECT P.idp, P.nombrep, V.ventas
  FROM Ventas V, Productos P
 WHERE V.idp=P.idp AND V.idubi=1 AND V.idhora=3 AND V.ventas > c
 ORDER BY V.ventas DESC
```

Por supuesto, este enfoque es mucho más rápido que la alternativa de calcular todas las ventas de productos y ordenarlas, pero hay algunos problemas importantes a resolver:

1. *¿Cómo se escoge el valor de corte de las ventas c ?* Se pueden emplear para este fin histogramas y otras estadísticas del sistema, pero puede resultar engañoso. Por un lado, las estadísticas que obtienen los SGBD sólo son aproximadas. Por otro, aunque se escoja un valor de corte que refleje con precisión los 10 primeros valores de ventas, puede que otras condiciones de la consulta eliminén parte de las tuplas seleccionadas y dejen menos de 10 tuplas en el resultado.
2. *¿Qué ocurre si se tienen más de 10 tuplas en el resultado?* Como la elección del valor de corte c es aproximada, se podrían obtener en el resultado más tuplas de las deseadas. Esto se soluciona fácilmente devolviendo al usuario sólo los 10 primeros. Se sigue ahorrando considerablemente con respecto al enfoque del cálculo de las ventas de todos los productos, gracias al podado conservador de la información de ventas irrelevante, mediante el valor de corte c .
3. *¿Qué ocurre si se tienen menos de 10 tuplas en el resultado?* Aunque se escoja de manera conservadora el valor de corte de ventas c , seguirían pudiéndose calcular menos de diez tuplas de resultados. En ese caso, se puede volver a ejecutar la consulta con un valor de corte menor, c_2 , o simplemente volver a ejecutar la consulta original sin valores de corte.

La efectividad del enfoque depende de la bondad de la estimación del valor de corte y, especialmente, de la minimización del número de veces que se obtienen menos tuplas de resultados que las deseadas.

16.5.2 Agregación en línea

Considérese la consulta siguiente, que determina el importe medio de las ventas por provincia:

```

SELECT      U.provincia, AVG (V.ventas)
FROM        Ventas V, Ubicaciones U
WHERE       V.idubi=U.idubi
GROUP BY    U.provincia

```

Esta consulta puede ser una consulta costosa si Ventas y Ubicaciones son relaciones grandes. No se pueden conseguir tiempos de respuesta rápidos con el enfoque tradicional de calcular la respuesta en su totalidad cuando se formula la consulta. Una alternativa, como hemos visto, es utilizar el cálculo previo. Otra alternativa es calcular la respuesta a la consulta cuando ésta se formule pero devolver al usuario una respuesta aproximada lo antes posible. A medida que avanza el cálculo, la calidad de la respuesta se va refinando continuamente. Este enfoque se denomina **agregación en línea**. Resulta muy atractivo para las consultas que incluyen agregación, ya que se dispone de técnicas eficientes para el cálculo y refinamiento de las respuestas aproximadas.

La agregación en línea se ilustra en la Figura 16.8. Para cada provincia —el criterio de

ESTADO	PRIORIZAR	Provincia	AVG(ventas)	Confianza	Intervalo
		Alicante	5.232,5	97%	103,4
		Almería	2.832,5	93%	132,2
		Cádiz	6.432,5	98%	52,3
		Zaragoza	4.243,5	92%	152,3

Figura 16.8 Agregación en línea

agrupamiento para la consulta de ejemplo— se muestra el valor actual del promedio de ventas, junto con un intervalo de confianza. La entrada para Lugo dice que la estimación actual de las ventas medias por tienda en Lugo es de 2.832,50 €, y que este valor se halla dentro del rango de 2.700,30 € a 2.964,70 € con una probabilidad del 93%. La barra de estado de la primera columna indica lo cerca que se está de llegar a un valor exacto para las ventas promedio, y la segunda columna indica si es una prioridad calcular las ventas promedio para esta provincia. La estimación de las ventas promedio para Lugo no es una prioridad, pero sí lo es hacerlo para Almería. Como indica la figura, el SGBD dedica más recursos del sistema a estimar las ventas promedio en las provincias de alta prioridad; la estimación para Arizona es mucho más precisa que para Lugo y tiene una probabilidad mayor. Los usuarios pueden definir la prioridad de cada provincia pulsando el botón Priorizar en cualquier momento de la ejecución. Este grado de interactividad, junto con la realimentación continua ofrecida por la interfaz gráfica, hace de la agregación en línea una técnica atractiva.

Para implementar la agregación en línea los SGBD deben incorporar técnicas estadísticas para proporcionar los intervalos de confianza de las respuestas aproximadas y utilizar **algoritmos no bloqueantes** para los operadores relacionales. Se dice que un algoritmo bloquea

Más allá de los árboles B+. Las consultas complejas han motivado añadir a los SGBD potentes técnicas de indexación. Además de los índices de árboles B+, Oracle 9i soporta los índices de mapas de bits y de reunión y los mantiene dinámicamente a medida que se actualizan las relaciones indexadas. Oracle 9i también soporta los índices para las expresiones sobre los valores de los atributos, como $10 * vent + plus$. SQL Server de Microsoft utiliza los índices de mapas de bits. IQ de Sybase soporta varios tipos de índices de mapas de bits, y puede que añada en breve soporte para los índices basados en la asociación lineal. UDS de Informix soporta los árboles R y XPS de Informix soporta los índices de mapas de bits.

si no produce tuplas de resultados hasta que ha consumido todas sus tuplas de entrada. Por ejemplo, el algoritmo de reunión por ordenación-mezcla bloquea porque la ordenación necesita todas las tuplas de entrada antes de determinar la primera tupla de resultados. La reunión de bucles anidados y la reunión por asociación son, por tanto, preferibles a la reunión por ordenación-mezcla para la agregación en línea. De manera parecida, la agregación basada en la asociación es mejor que la agregación basada en la ordenación.

16.6 TÉCNICAS DE IMPLEMENTACIÓN PARA OLAP

En este apartado se resumen algunas técnicas de implementación motivadas por el entorno OLAP. El objetivo es proporcionar una comprensión del modo en que los sistemas OLAP se diferencian de los sistemas SQL más tradicionales; este análisis dista mucho de ser exhaustivo.

El entorno principalmente de lectura de los sistemas OLAP hace despreciable la sobrecarga de la CPU por el mantenimiento de los índices, mientras que la exigencia de tiempos de respuesta interactivos para las consultas en conjuntos de datos de tamaño muy grande vuelve muy importante la disponibilidad de índices adecuados. Esta combinación de factores ha llevado al desarrollo de nuevas técnicas de indexación. A continuación se estudiarán varias de esas técnicas. Despues se tomará en consideración brevemente los modos de organización de los archivos y otros aspectos de la implementación de OLAP.

Cabe destacar que el énfasis en el procesamiento de consultas y en las aplicaciones de ayuda a la toma de decisiones en los sistemas OLAP se está complementando con un mayor énfasis en la evaluación de consultas SQL complejas en los sistemas de SQL tradicionales. Los sistemas de SQL tradicionales están evolucionando para soportar consultas de estilo OLAP de manera más eficiente, permitiendo nuevas estructuras (por ejemplo, CUBE y las funciones de ventana) e incorporando técnicas de implementación que antes sólo se podían encontrar en los sistemas OLAP especializados.

16.6.1 Índices de mapas de bits

Considérese una tabla que describe a los clientes:

Clientes(*idcli: integer*, *nombre: string*, *sexo: boolean*, *categoría: integer*)

El valor de *categoría* es un entero entre 1 y 5, y sólamente se registran dos valores para *sexo*. Las columnas con pocos valores posibles se denominan **dispersas**. La dispersión se puede aprovechar para crear una nueva clase de índice que acelere enormemente las consultas a esas columnas.

La idea es registrar los valores de las columnas dispersas como secuencias de bits, una por cada valor posible. Por ejemplo, el valor de *sexo* es 10 o 01; un 1 en la primera posición denota masculino, mientras que un 1 en la segunda posición denota femenino. De manera similar, 10000 denota el valor de *categoría* 1, mientras que 00001 denota el valor 5 de *categoría*.

Si se consideran los valores de *sexo* de todas las filas de la tabla Clientes, se pueden tratar como un conjunto de dos **vectores de bits**, uno de los cuales tiene el valor asociado M(asculino) y el otro el valor asociado F(emenino). Cada vector de bits tiene un bit por fila de la tabla Clientes, que indica si el valor de esa fila es el valor asociado con el vector de bits. El conjunto de vectores de bits de una columna se denomina **índice del mapa de bits** de esa columna.

Se puede ver un ejemplar de la tabla Clientes, junto con los índices de mapas de bits de *sexo* y *categoría*, en la Figura 16.9.

M	F
1	0
1	0
0	1
1	0

<i>idcli</i>	<i>nombre</i>	<i>sexo</i>	<i>categoría</i>
112	José	M	3
115	Ramón	M	5
119	Susana	F	5
112	Guillermo	M	4

1	2	3	4	5
0	0	1	0	0
0	0	0	0	1
0	0	0	0	1
0	0	0	1	0

Figura 16.9 Índices de mapas de bits de la relación Clientes

Los índices de mapas de bits ofrecen dos ventajas importantes respecto a los índices convencionales de asociación y de árboles. En primer lugar, permiten el empleo de operaciones de bits eficientes para responder las consultas. Por ejemplo, considérese la consulta, “¿Cuántos clientes varones tienen categoría 5?” Se puede tomar el primer vector de bits de *sexo* y realizar una operación AND bit a bit con el quinto vector de bits de *categoría* para obtener un vector de bits que tenga un 1 para cada cliente varón con categoría 5. Luego se puede contar el número de unos de ese vector de bits para responder la consulta. En segundo lugar, los índices de mapas de bits pueden ser mucho más compactos que los índices de árbol B+ tradicionales, y se prestan al empleo de técnicas de compresión.

Los vectores de bits se corresponden estrechamente con las listas idr empleadas para representar entradas de datos en la Alternativa (3) a los índices de árbol B+ tradicionales (véase el Apartado 9.2). De hecho, los vectores de bits para un valor dado de *edad* se pueden considerar, por ejemplo, como una representación alternativa de la lista idr de ese valor.

Esto sugiere una manera de combinar los vectores de bits (y sus ventajas de procesamiento bit a bit) con los índices de árboles B+: se puede utilizar la Alternativa (3) para las entradas de datos, empleando una representación de vector de bits de las listas idr. Hay que tener en cuenta, no obstante, que si una lista idr es muy pequeña es posible que la representación de vectores de bits sea mucho mayor que la lista de valores idr, aunque el vector de bits esté comprimido. Además, el empleo de la compresión provoca costes de descompresión, que equilibran algunas de las ventajas de cálculo de la representación de vectores de bits.

Un enfoque más flexible es emplear una representación estándar de la lista idr para algunos valores de la clave (intuitivamente, para aquellos que contengan pocos elementos) y una representación de vectores de bits para otros valores de la clave (los que contengan muchos elementos y, por tanto, se presten a una representación compacta de vectores de bits).

Este enfoque híbrido, que se puede adaptar fácilmente para que funcione con los índices asociativos y con los de árboles B+, tiene tanto ventajas como inconvenientes en relación con el enfoque estándar de las listas de rids:

1. Se puede aplicar incluso a columnas que no sean dispersas; es decir, en las que puedan aparecer muchos valores. Los niveles de índices (o el esquema de asociación) permite hallar rápidamente la “lista” de rids, en una lista estándar o en una representación de vectores de bits, para un valor dado de la clave.
2. En general, el índice es más compacto, ya que se puede emplear la representación de vectores de bits para las listas idr largas. También se tienen las ventajas del procesamiento rápido de los vectores de bits.
3. Por otro lado, la representación de vectores de bits de una lista idr se basa en una correspondencia desde una posición del vector a un idr. (Esto es cierto para cualquier representación de vectores de bits, no sólo para este enfoque híbrido.) Si el conjunto de filas es estático, y no nos preocupamos por las inserciones y eliminaciones de filas, resulta sencillo garantizar esto mediante la asignación de rids contiguos para las filas de la tabla. Si hay que permitir inserciones y eliminaciones, todo se complica. Por ejemplo, se pueden seguir asignando rids de manera contigua tabla por tabla y limitarnos a registrar los rids que corresponden a las filas eliminadas. Ahora los vectores de bits pueden ser más largos que el número de filas actual y hace falta una reorganización periódica para compactar los “agujeros” de la asignación de rids.

16.6.2 Índices de reunión

El cálculo de reuniones con tiempos de respuesta cortos resulta extremadamente difícil para las relaciones de tamaño muy grande. Un enfoque de este problema es crear un índice diseñado para acelerar consultas de reunión concretas. Supóngase que la tabla Clientes se debe reunir con una tabla denominada Adquisiciones (que registra las compras hechas por los clientes) según el campo *idcli*. Se puede crear un conjunto de $\langle c, a \rangle$ pares, donde *a* es el idr del registro de Adquisiciones que se reúne con el registro de Clientes con *idcli* *c*.

Se puede generalizar esta idea para que soporte las reuniones entre más de dos relaciones. Se analizará el caso especial de un esquema en estrella, en el que es posible que la tabla de hechos se reúna con varias tablas de dimensiones. Considérese una consulta de reunión que reúna la tabla de hechos H con las tablas de dimensiones D1 y D2 e incluya condiciones de selección para la columna *C*₁ de la tabla D1 y la columna *C*₂ de la tabla D2. Se guarda la tupla $\langle r_1, r_2, r \rangle$ del índice de reunión si *r*₁ es el idr de la tupla de la tabla D1 con valor *c*₁ de la columna *C*₁, *r*₂ es el idr de la tupla de la tabla D2 con valor *c*₂ de la columna *C*₂, *r* es el idr de una tupla de la tabla de hechos F y estas tres tuplas se reúnen entre sí.

El inconveniente de los índices de reunión es que el número de índices puede aumentar rápidamente si varias columnas de cada tabla de dimensiones están implicadas en selecciones y

Consultas complejas. El optimizador de DB2 de IBM reconoce las consultas de reunión en estrella y lleva a cabo semirreuniones basadas en idr (empleando filtros Bloom) para filtrar la tabla de hechos. Luego se vuelven a reunir las filas de la tabla de hechos con las tablas de dimensiones. Las consultas de dimensiones complejas (multitablas) (denominadas *consultas copos de nieve*) están soportadas. DB2 permite también que CUBE utilice algoritmos inteligentes que minimicen las ordenaciones. SQL Server de Microsoft optimiza mucho las consultas de reunión en estrella. Considera la toma del producto vectorial de las tablas de dimensiones de pequeño tamaño con la tabla de hechos, el empleo de índices de reunión y las semirreuniones basadas en idr. Oracle 9i también permite que los usuarios creen dimensiones para declarar jerarquías y dependencias funcionales. Soporta el operador CUBE y optimiza las consultas de reunión en estrella mediante la eliminación de las reuniones cuando ninguna columna de una tabla de dimensiones forma parte del resultado de la consulta. También se han desarrollado productos de SGBD de manera específica para las aplicaciones de ayuda a la toma de decisiones, como IQ de Sybase.

reuniones con la tabla de hechos. Un tipo alternativo de índice de reunión evita este problema. Considérese el ejemplo con la tabla de hechos H y las tablas de dimensiones D1 y D2. Sea C_1 una columna de D1 en la que se expresa una selección de alguna consulta que reúne D1 con H. Conceptualmente, ahora se reúne H con D1 para ampliar los campos de H con los de D1 e indexar H de acuerdo con el “campo virtual” C_1 : si la tupla de D1 de valor c_1 de la columna C_1 se reúne con la tupla de H con idr r , se añade la tupla $\langle c_1, r \rangle$ al índice de reunión. Se crea una reunión de este tipo para cada columna de D1 o de D2 que implique una selección de alguna reunión con H; C_1 es un ejemplo de ese tipo de columna.

El precio pagado con respecto a la versión anterior de los índices de reunión es que los índices de reunión creados de esta manera se tienen que combinar (intersección idr) para tratar con las consultas de reunión de interés. Esto se puede lograr de manera eficiente si se hace que los índices nuevos sean índices de *mapas de bits*; el resultado se denomina **índice de reunión de mapas de bits**. Esta idea funciona especialmente bien si columnas como C_1 son dispersas y, por tanto, idóneas para la indexación de mapas de bits.

16.6.3 Organizaciones de archivos

Dado que muchas consultas OLAP sólo afectan a unas cuantas columnas de una relación de gran tamaño, el particionamiento vertical se vuelve atractivo. Sin embargo, guardar las relaciones columna a columna puede degradar el rendimiento de las consultas que afecten a varias columnas. Una alternativa en un entorno donde la lectura es la operación más habitual es guardar la relación fila a fila, pero guardando también cada columna por separado.

Una organización de archivos más radical es considerar la tabla de hechos como un array multidimensional de gran tamaño y guardarla e indexarla como tal. Este enfoque se adopta en los sistemas MOLAP. Dado que el array es mucho más grande que la memoria principal disponible, se descompone en fragmentos contiguos, como se analiza en el Apartado 15.8. Además,

se crean índices B+ tradicionales para permitir la rápida recuperación de los fragmentos que contengan tuplas con valores en un rango dado para una o más dimensiones.

16.7 ALMACENES DE DATOS

Los almacenes de datos contienen datos combinados de muchas fuentes, junto con información de resumen, y abarcan un largo periodo de tiempo. Los almacenes son mucho más grandes que otros tipos de bases de datos; son frecuentes tamaños de entre varios gigabytes y terabytes. Las cargas de trabajo típicas incluyen consultas *ad hoc* bastante complejas y son importantes los tiempos de respuesta rápidos. Estas características diferencian las aplicaciones de los almacenes de las aplicaciones OLTP, y se deben emplear diseños de SGBD y técnicas de implementación diferentes para conseguir resultados satisfactorios. Se necesitan SGBD distribuidos con buena escalabilidad y elevada disponibilidad (que se obtiene guardando las tablas de manera redundante en más de un sitio) para los almacenes muy grandes.

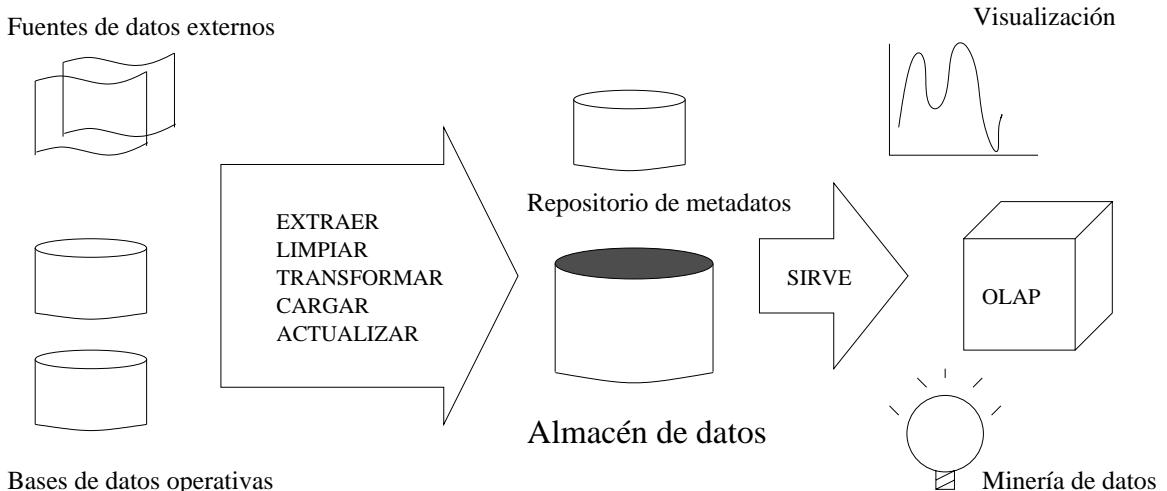


Figura 16.10 Arquitectura típica de los almacenes de datos

En la Figura 16.10 se puede ver una arquitectura de almacenamiento típica. Las operaciones diarias de cualquier organización tienen acceso a las **bases de datos operativas** y las modifican. Los datos de esas bases de datos operativas y de otras fuentes externas (por ejemplo, los perfiles de consumidores proporcionados por consultores externos) se **extraen** mediante interfaces como JDBC (véase el Apartado 6.2).

16.7.1 Creación y mantenimiento de almacenes de datos

Hay que afrontar muchos desafíos a la hora de crear y mantener almacenes de datos de gran tamaño. Se debe diseñar un buen esquema de base de datos para guardar un conjunto integrado de datos copiados de diversas fuentes. Por ejemplo, puede que el almacén de una empresa incluya las bases de datos de los departamentos de inventario y de personal, junto

con las bases de datos de las ventas conservadas por las oficinas de diferentes países. Dado que a menudo las bases de datos las crean y mantienen grupos diferentes, hay varias incoherencias semánticas entre unas bases de datos y otras, como pueden ser las diferentes unidades monetarias, los nombres diferentes para un mismo atributo y diferencias en la manera de normalizar o estructurar las tablas; estas diferencias se deben reconciliar cuando se llevan los datos al almacén. Una vez diseñado el esquema del almacén, éste se debe llenar y, a lo largo del tiempo, se debe mantener consistente con las bases de datos de origen.

Los datos se **extraen** de las bases de datos operativas y de fuentes externas, se **limpien** para minimizar errores, se rellena la información que falta siempre que resulta posible y se **transforman** para reconciliar las incoherencias semánticas. La transformación de los datos se suele conseguir mediante la definición de una vista relacional de las tablas en los orígenes de datos (las bases de datos operativas y otras fuentes externas). La **carga** de los datos consiste en materializar esas vistas y guardarlas en el almacén. A diferencia de las vistas estándar de los SGBD relacionales, por tanto, la vista se guarda en una base de datos (el almacén) que es diferente de la(s) base(s) de datos que contiene(n) las tablas sobre las que esté definida.

Los datos limpiados y transformados se **cargan** finalmente en el almacén. En esta etapa se lleva a cabo un preprocesamiento adicional como la ordenación y la generación de información de resumen. Se partitionan los datos y se crean índices en aras de la eficiencia. Debido al gran volumen de datos, la carga es un proceso lento. La carga de un terabyte de datos de manera secuencial puede tardar semanas e, incluso, la carga de un gigabyte puede tardar horas. Por tanto, el paralelismo resulta importante para la carga de los almacenes de datos.

Una vez cargados los datos en el almacén, se deben adoptar medidas adicionales para garantizar que se **actualicen** de manera periódica para que reflejen las actualizaciones de las fuentes de datos y **purguen** periódicamente los datos viejos (quizás a medios de archivo). Obsérvese la conexión entre el problema de la actualización de las tablas de los almacenes y el mantenimiento asíncrono de réplicas de tablas en los SGBD distribuidos. El mantenimiento de réplicas de las relaciones de origen es una parte esencial del almacenamiento, y este dominio de aplicación es un factor importante de la popularidad de las réplicas asíncronas, aunque la réplica asíncrona viole el principio de independencia de los datos distribuidos. El problema de la actualización de las tablas de los almacenes (que son vistas materializadas de tablas de las bases de datos de origen) también ha renovado el interés en el mantenimiento incremental de las vistas materializadas. (Las vistas materializadas se estudian en el Apartado 16.8.)

Una tarea importante del mantenimiento de un almacén es el seguimiento de los datos que se almacenan en él; esta contabilidad se lleva a cabo guardando información sobre los datos del almacén en los catálogos del sistema. Los catálogos del sistema asociados con un almacén son muy grandes y se suelen guardar y gestionar en una base de datos diferente denominada **repositorio de metadatos**. El tamaño y la complejidad de los catálogos se deben en parte al tamaño y complejidad del propio almacén y en parte debido a que hay que conservar gran cantidad de información administrativa. Por ejemplo, hay que realizar un seguimiento del origen de cada tabla del almacén y de cuándo se actualizó por última vez, además de describir sus campos.

El valor de un almacén depende en última instancia del análisis que permite. Se suele tener acceso a los datos de los almacenes y analizarlos mediante gran variedad de herramientas, como los motores de consulta OLAP, los algoritmos de minería de datos, las herramientas de visualización de la información, los paquetes estadísticos y los generadores de informes.

16.8 VISTAS Y AYUDA A LA TOMA DE DECISIONES

Las vistas se emplean mucho en las aplicaciones de ayuda a la toma de decisiones. Los diferentes grupos de analistas de una misma organización suelen estar interesados por aspectos diferentes del negocio, y resulta conveniente definir vistas que den a cada grupo una perspectiva de los detalles del negocio que le interesan. Una vez definida la vista, se pueden escribir consultas o nuevas definiciones de vistas que la utilicen, como se vio en el Apartado 3.6; a este respecto una vista es exactamente igual que una tabla básica. La evaluación de las consultas formuladas a las vistas es muy importante para las aplicaciones de ayuda a la toma de decisiones. En este apartado se considerará el modo en que se pueden evaluar de manera eficiente esas consultas tras ubicar las vistas en el contexto de las aplicaciones de ayuda a la toma de decisiones.

16.8.1 Vistas, OLAP y almacenamiento

Las vistas están estrechamente relacionadas con OLAP y con el almacenamiento de datos.

Las consultas OLAP suelen ser consultas de agregación. Los analistas desean respuestas rápidas a estas consultas a conjuntos de datos de tamaño muy grande, y es natural que se consideren las vistas calculadas previamente (véanse los Apartados 16.9 y 16.10). En especial, el operador CUBE —que se estudia en el Apartado 16.3— da lugar a varias consultas de agregación que se hallan estrechamente relacionadas. Las relaciones que existen entre las muchas consultas de agregación que surgen de una sola operación CUBE se pueden aprovechar para desarrollar estrategias de cálculo previo muy efectivas. La idea es escoger un subconjunto de las consultas de agregación para materializarlas de modo que las consultas CUBE típicas se puedan responder rápidamente mediante el empleo de las vistas materializadas y la realización de algún cálculo adicional. La elección de las vistas que se deben materializar viene influída por el número de consultas que puedan llegar a acelerar y por la cantidad de espacio necesario para guardar cada vista materializada (dado que hay que trabajar con una cantidad dada de espacio de almacenamiento).

Los almacenes de datos no son más que conjuntos de tablas replicadas asíncronamente y de vistas sincronizadas periódicamente. Cada almacén se caracteriza por su tamaño, el número de tablas implicadas y el hecho de que la mayor parte de las tablas subyacentes provienen de bases de datos externas mantenidas de manera independiente. No obstante, el problema fundamental del mantenimiento de los almacenes es el mantenimiento asíncrono de las tablas replicadas y de las vistas materializadas (véase el Apartado 16.10).

16.8.2 Consultas sobre vistas

Considérese la vista siguiente, VentasRegionales, que calcula las ventas de productos por categoría y provincia:

```
CREATE VIEW VentasRegionales (categoría, ventas, provincia)
AS SELECT P.categoría, V.ventas, U.provincia
      FROM Productos P, Ventas V, Ubicaciones U
     WHERE P.idp = V.idp AND V.idubi = U.idubi
```

La consulta siguiente calcula las ventas totales de cada categoría por provincia:

```
SELECT R.categoría, R.provincia, SUM (R.ventas)
FROM VentasRegionales R
GROUP BY V.categoría, V.provincia
```

Aunque la norma de SQL no especifica el modo de evaluar las consultas sobre vistas, resulta útil pensar en términos de un proceso denominado **modificación de consultas**. La idea es sustituir la presencia de VentasRegionales en la consulta por la definición de la vista. El resultado de esta consulta es

```
SELECT R.categoría, R.provincia, SUM (R.ventas)
FROM ( SELECT P.categoría, V.ventas, U.provincia
       FROM Productos P, Ventas V, Ubicaciones U
      WHERE P.idp = V.idp AND V.idubi = U.idubi ) AS R
GROUP BY R.categoría, R.provincia
```

16.9 MATERIALIZACIÓN DE VISTAS

Las consultas sobre vistas se pueden responder mediante la técnica de modificación de consultas que se acaba de describir. A menudo, sin embargo, hay que responder muy rápidamente consultas sobre definiciones de vistas complejas debido a que los usuarios implicados en las actividades de ayuda a la toma de decisiones necesitan tiempos de respuesta interactivos. Incluso con técnicas de optimización y evaluación sofisticadas hay un límite a la rapidez con la que se pueden responder esas preguntas. Además, si las tablas subyacentes se hallan en una base de datos remota, puede que el enfoque de modificación de la consulta ni siquiera sea factible debido a problemas como la conectividad y la disponibilidad.

Una alternativa a la modificación de consultas es calcular previamente la definición de la vista y guardar el resultado. Cuando se formula una consulta a la vista se ejecuta la consulta (sin modificar) directamente sobre el resultado calculado previamente. Es probable que este enfoque, denominado **materialización de vistas**, sea mucho más rápido que el de modificación de consultas, ya que no hace falta evaluar la vista compleja al calcular la consulta. Las vistas materializadas se pueden utilizar durante el procesamiento de consultas del mismo modo que las relaciones normales; por ejemplo, se pueden crear índices de las vistas materializadas para acelerar aún más el procesamiento de las consultas. El inconveniente, por supuesto, es que hay que conservar la consistencia de la vista calculada previamente (*o materializada*) siempre que se actualicen las tablas subyacentes.

16.9.1 Problemas de la materialización de vistas

Se deben considerar tres cuestiones en relación con la materialización de las vistas:

1. ¿Qué vistas se deben materializar y qué índices se deben crear para las vistas materializadas?
2. Dada una consulta sobre una vista y un conjunto de vistas materializadas, ¿se pueden aprovechar las vistas materializadas para responder la consulta?

3. ¿Cómo se deben sincronizar las vistas materializadas con las modificaciones de las tablas subyacentes? La elección de la técnica de sincronización depende de varios factores, como si las tablas subyacentes se hallan en una base de datos remota. Este problema se estudia en el Apartado 16.10.

Las respuestas a las dos primeras preguntas están relacionadas. La elección de las vistas que se deben materializar y del índice viene determinada por la carga de trabajo esperada, y el estudio de la indexación del Capítulo 13 también es relevante para esta cuestión. Sin embargo, la elección de las vistas que se deben materializar es más compleja que la mera elección de índices para un conjunto de tablas de la base de datos, ya que el rango de vistas alternativas que materializar es más amplio. El objetivo es materializar un conjunto pequeño, cuidadosamente escogido, de vistas que se puedan utilizar para responder rápidamente la mayor parte de las consultas importantes. Recíprocamente, una vez escogido un conjunto de vistas a materializar, hay que considerar cómo se pueden utilizar para responder una consulta dada.

Considérese la vista VentasRegionales. Comprende una reunión de Ventas, Productos y Ubicaciones y es probable que resulte costoso de calcular. Por otro lado, si se materializa y se guarda con un índice de árbol B+ agrupado para la clave de búsqueda *(categoría, provincia, ventas)*, se puede responder la consulta de ejemplo mediante una exploración que sólo afecte al índice.

Dada la vista materializada y este índice, también se pueden responder consultas con eficiencia de la manera siguiente:

```
SELECT R.provincia, SUM (R.ventas)
FROM VentasRegionales R
WHERE R.categoría = 'Portátil'
GROUP BY R.provincia
```

Para responder esta consulta se puede utilizar el índice de la vista materializada para hallar la primera entrada de hoja del índice con *categoría* = 'Portátil' y examinar luego el nivel de la hoja hasta llegar a la primera entrada con *categoría* diferente de Portátil.

El índice dado es menos efectivo para la entrada siguiente, para la cual es necesario explorar todo el nivel de la hoja:

```
SELECT R.provincia, SUM (R.ventas)
FROM VentasRegionales R
WHERE R.provincia = 'Coruña'
GROUP BY R.categoría
```

Este ejemplo indica el modo en que la elección de las vistas que se deben materializar y de los índices que hay que crear se ve afectada por la carga de trabajo esperada. Este punto queda ilustrado por el ejemplo siguiente.

Considérense las dos consultas siguientes:

```
SELECT P.categoría, SUM (V.ventas)
FROM Productos P, Ventas V
WHERE P.idp = V.idp
GROUP BY P.categoría
```

```

SELECT      U.provincia, SUM (V.ventas)
FROM        Ubicaciones U, Ventas V
WHERE       U.idubi = V.idubi
GROUP BY   U.provincia

```

Estas dos consultas necesitan la reunión la tabla Ventas (que es probable que sea muy grande) con otra tabla y la agregación del resultado. ¿Cómo se puede utilizar la materialización para acelerar estas consultas? El enfoque más sencillo es calcular previamente cada una de las reuniones implicadas (Productos con Ventas y Ubicaciones con Ventas) o calcular previamente cada consulta en su totalidad. Un enfoque alternativo es definir la vista siguiente:

```

CREATE      VIEW VentasTotales (idp, idubi, total)
AS SELECT   V.idp, V.idubi, SUM (V.ventas)
           FROM     Ventas V
           GROUP BY V.idp, V.idubi

```

Se puede materializar la vista VentasTotales y utilizarla en lugar de Ventas en las dos consultas de ejemplo:

```

SELECT      P.categoría, SUM (T.total)
FROM        Productos P, VentasTotales T
WHERE       P.idp = T.idp
GROUP BY   P.categoría

```

```

SELECT      U.provincia, SUM (T.total)
FROM        Ubicaciones U, VentasTotales T
WHERE       U.idubi = T.idubi
GROUP BY   U.provincia

```

16.10 MANTENIMIENTO DE VISTAS MATERIALIZADAS

Se dice que una vista materializada se **actualiza** cuando se hace consistente con las modificaciones de sus tablas subyacentes. El proceso de actualización de una vista para mantenerla consistente con las modificaciones de las tablas subyacentes se suele conocer como **mantenimiento de la vista**. Hay que considerar dos cuestiones

1. *¿Cómo* actualizar una vista cuando se modifica alguna tabla subyacente? Dos aspectos de especial interés son el modo de mantener las vistas *de manera incremental*, es decir, sin volver a calcular desde el principio cuando se producen modificaciones en las tablas subyacentes; y la manera de mantener las vistas en entornos distribuidos como los almacenes de datos.
2. *¿Cuándo* se debe actualizar una vista en respuesta a las modificaciones en las tablas subyacentes?

16.10.1 Mantenimiento incremental de vistas

Un enfoque sencillo de la actualización de vistas es limitarse a volver a calcular la vista cuando se modifica alguna tabla subyacente. De hecho, puede que sea una estrategia razonable en algunos casos. Por ejemplo, si las tablas subyacentes se hallan en una base de datos remota, se puede volver a calcular la vista de manera periódica y enviarla al almacén de datos donde se materializa. Esto tiene la ventaja de que no hace falta replicar las tablas subyacentes en el almacén.

No obstante, siempre que sea posible los algoritmos para la actualización de las vistas deben ser **incrementales**, en el sentido de que el coste sea proporcional a la amplitud de la modificación, más que al coste de volver a calcular la vista desde el principio.

Para comprender la intuición subyacente a los algoritmos para el mantenimiento incremental de vistas hay que observar que cada fila de la vista materializada puede aparecer varias veces, según la frecuencia con que se obtuvo. (Recuérdese que los valores duplicados no se eliminan del resultado de las consultas SQL a menos que se emplee la cláusula DISTINCT.) En este apartado se estudiará la semántica multiconjunto, aunque se emplee notación propia del álgebra relacional.) La idea principal tras los algoritmos de mantenimiento incremental es calcular de manera eficiente las modificaciones de las filas de la vista, tanto las filas nuevas como las modificaciones del contador asociado a una fila; si el contador de la fila pasa a ser 0, esa fila se elimina de la vista.

Se presentará un algoritmo de mantenimiento incremental para las vistas definidas mediante proyección, reunión binaria y agregación; se tratarán estas operaciones porque ilustran las ideas principales. El enfoque se puede ampliar a otras operaciones como la selección, la unión, la intersección y la diferencia (multiconjuntos), así como a las expresiones que contengan varios operadores. La idea principal sigue siendo conservar el número de veces que se obtiene cada fila de la vista, pero los detalles del modo de calcular de manera eficiente las modificaciones de las filas de la vista y de los contadores asociados difieren.

Vistas de proyección

Considérese la vista V definida en términos de una proyección sobre la tabla R ; es decir, $V = \pi(R)$. Cada fila v de V tiene un contador asociado, correspondiente al número de veces que se puede obtener, que es el número de filas de R que dan como resultado v al aplicar la proyección. Supóngase que se modifica R mediante la inserción del conjunto de filas R_i y se elimina el conjunto de filas ya existentes R_e ¹. Se calcula $\pi(R_i)$ y se añade a V . Si el multiconjunto $\pi(R_i)$ contiene la fila r con el contador c y r no aparece en V , se añade a V con el contador c . Si r se halla en V , se añade c a su contador. También se calcula $\pi(R_e)$ y se resta de V . Obsérvese que, si r aparece en $\pi(R_e)$ con el contador c , también debe aparecer en V con un contador más elevado²; se resta c del contador de r en V .

Por ejemplo, considérese la vista $\pi_{ventas}(Ventas)$ y la instancia de Ventas de la Figura 16.2. Cada fila de la vista tiene una sola columna; el valor 25 (la fila con ese valor) aparece

¹Estos conjuntos pueden ser multiconjuntos de filas. Por simplicidad, se puede tratar la modificación de una fila como una inserción seguida de una eliminación.

²Como ejercicio sencillo, considérese el motivo de que esto deba ser así.

con contador 1, y el valor 10 aparece con contador 3. Si se elimina una de las filas de Ventas con *ventas* igual a 10, el contador de (la fila con) valor 10 de la vista pasa a 2. Si se inserta una fila nueva en Ventas con *ventas* igual a 99, la vista tendrá una fila con valor 99.

Un punto importante es que hay que mantener las cuentas asociadas con las filas aunque la definición de la vista emplee la cláusula DISTINCT, lo que significa que los duplicados se eliminan de la vista. Considérese la misma vista con la semántica de conjunto —se emplea la cláusula DISTINCT en la definición de la vista en SQL— y supóngase que se elimina una de las filas de Ventas con *ventas* igual a 10. ¿Sigue teniendo la vista alguna fila con valor 10? Para determinar que la respuesta es afirmativa hay que conservar los contadores de las filas, aunque cada fila (con contador distinto de cero) sólo aparezca una vez en la vista materializada.

Vistas de reunión

Considérese a continuación la vista V definida como reunión de dos tablas, $R \bowtie S$. Supóngase que se modifica R mediante la inserción del conjunto de filas R_i y la eliminación del conjunto de filas R_e . Se calcula $R_i \bowtie S$ y se añade el resultado a V . También se calcula $R_e \bowtie S$ y se resta el resultado de V . Obsérvese que, si r aparece en $R_e \bowtie S$ con el contador c , también debe aparecer en V con un contador más elevado³.

Vistas con agregación

Considérese la vista V definida sobre R mediante GROUP BY para la columna G y una operación de agregación sobre la columna A . Cada fila v de la vista resume un grupo de tuplas de R y es de la forma $\langle g, resumen \rangle$, donde g es el valor de la columna de agrupación G y la información de resumen depende de la operación de agregación. Para mantener esa vista de manera incremental, en general hay que tener un resumen más detallado que la mera información incluida en la vista. Si la operación de agregación es COUNT, sólo hace falta mantener un contador c para cada fila v de la vista. Si se inserta una fila r en R , y no hay ninguna fila v de V con $v.G = r.G$, se añade una fila nueva $\langle r.G, 1 \rangle$. Si hay una fila v con $v.G = r.G$, se incrementa su contador. Si se elimina la fila r de R , se disminuye el contador de la fila v con $v.G = r.G$; v se puede eliminar si su contador llega a 0, ya que, entonces, se ha eliminado de R la última fila de ese grupo.

Si la operación de agregación es SUM hay que mantener una suma s y también un contador c . Si se inserta una fila r en R y no hay ninguna fila v de V con $v.G = r.G$, se añade una fila nueva $\langle r.G, a, 1 \rangle$. Si hay una fila $\langle r.G, s, c \rangle$, se sustituye por $\langle r.G, s + a, c + 1 \rangle$. Si se elimina la fila r de R , se sustituye la fila $\langle r.G, s, c \rangle$ por $\langle r.G, s - a, c - 1 \rangle$; se puede eliminar v si su contador llega a 0. Obsérvese que, sin el contador, no se sabe cuándo se debe eliminar v , ya que la suma de un grupo puede ser 0 aunque el grupo contenga alguna fila.

Si la operación de agregación es AVG, hay que mantener una suma s , un contador c y el promedio de cada fila de la vista. La suma y el contador se mantienen de manera incremental, como ya se ha descrito, y el promedio se calcula como s/c .

Las operaciones de agregación MIN y MAX pueden llegar a ser costosas de mantener. Considérese MIN. Para cada grupo de R se mantienen $\langle g, m, c \rangle$, donde m es el valor mínimo de la

³Como un ejercicio sencillo más, considérese el motivo de que esto deba ser así.

columna A del grupo g y c es el contador del número de filas r de R con $r.G = g$ y $r.A = m$. Si se inserta la fila r en R y $r.G = g$, si $r.A$ es mayor que el mínimo m del grupo g , se puede ignorar r . Si $r.A$ es igual al mínimo m del grupo de r , se sustituye la fila de resumen del grupo por $\langle g, m, c + 1 \rangle$. Si $r.A$ es menor que el mínimo m del grupo de r , se sustituye el resumen del grupo por $\langle g, r.A, 1 \rangle$. Si se elimina la fila r de R y $r.A$ es igual al mínimo m del grupo de r , entonces hay que disminuir el contador del grupo. Si el contador es mayor que 0, basta con sustituir el resumen del grupo por $\langle g, m, c - 1 \rangle$. Sin embargo, si el contador llega a 0, eso significa que la fila con el mínimo registrado de valor A se ha eliminado de R y hay que recuperar el valor más pequeño de A entre las filas restantes de R con el valor de grupo $r.G$ —y puede que esto exija la recuperación de todas las filas de R con el valor de grupo $r.G$ —.

16.10.2 Mantenimiento de las vistas en almacenes de datos

Las vistas materializadas en los almacenes de datos se pueden basar en tablas fuente de bases de datos remotas. Las técnicas de réplica asíncrona permiten comunicar al almacén las modificaciones de la fuente, pero la actualización incremental de las vistas en un entorno distribuido presenta algunos retos únicos. Para ilustrarlo, se considerará una vista simple que identifica los suministradores de juguetes.

```
CREATE VIEW SuministradoresJuguetes (ids)
AS SELECT S.ids
        FROM Suministradores S, Productos P
        WHERE S.idp = P.idp AND P.categoría = 'Juguetes'
```

Suministradores es una tabla nueva introducida para este ejemplo; supóngase que sólo contiene dos campos, *ids* e *idc*, que indican que el suministrador *ids* proporciona el componente *idc*. La ubicación de las tablas *Productos* y *Suministradores* y de la vista *SuministradoresJuguetes* influye en el modo en que se mantiene la vista. Supóngase que las tres se mantienen en un mismo sitio. Se puede mantener la vista de manera incremental mediante las técnicas examinadas en el Apartado 16.10.1. Si se crea una réplica de la vista en otro sitio, se pueden controlar las modificaciones de la vista materializada y aplicarlos en ese segundo sitio mediante técnicas de réplica asíncrona.

Pero ¿qué pasa si *Productos* y *Suministradores* se hallan en un sitio y la vista (sólo) se materializa en un segundo lugar? Para justificar esta situación hay que observar que, si se utiliza el primer sitio para los datos operativos y el segundo soporta el análisis complejo, puede que los dos sitios estén administrados por grupos diferentes. La opción de materializar *SuministradoresJuguetes* (una vista de interés para el segundo grupo) en el primer sitio (administrado por un grupo diferente) no resulta atractiva, y puede que ni siquiera sea posible; puede que los administradores del primer sitio no deseen tratar con las vistas de otras personas, y puede que los administradores del segundo sitio no deseen coordinarse con otros siempre que modifiquen las definiciones de las vistas. Como justificación adicional de la materialización de vistas en una ubicación diferente de la de las tablas fuente, obsérvese que puede que *Productos* y *Suministradores* se hallen en dos sitios diferentes. Aunque se materialice *SuministradoresJuguetes* en uno de esos sitios, una de las dos tablas seguirá siendo remota.

Ahora que se ha presentado la justificación para el mantenimiento de *SuministradoresJuguetes* en una ubicación (llamémoslo Almacén) diferente del que (llamémoslo Fuente) contiene

Productos y Suministradores, considérense las dificultades planteadas por la distribución de los datos. Supóngase que se inserta un nuevo registro de Productos (con la *categoría* = ‘Juguetes’). Se puede intentar mantener la vista de manera incremental del modo siguiente:

1. El sitio Almacén envía esta actualización al sitio Fuente.
2. Para actualizar la vista hace falta comprobar la tabla Suministradores para hallar los suministradores del elemento y, por tanto, el sitio Almacén pide esa información al sitio Fuente.
3. El sitio Fuente devuelve el conjunto de suministradores del elemento vendido, y el sitio Almacén actualiza la vista de manera incremental.

Esto funciona cuando no hay ninguna modificación más en el sitio Fuente entre los pasos (1) y (3). Si hay modificaciones, sin embargo, la vista materializada puede dejar de ser correcta —reflejando un estado que no puede surgir nunca salvo por las anomalías introducidas por el algoritmo de actualización anterior, incremental e ingenuo—. Para verlo, supóngase que Productos está vacía y que Suministradores contiene inicialmente sólo la fila $\langle s1, 5 \rangle$ y considérese la secuencia de eventos siguiente:

1. Se inserta el producto de $idp = 5$ con la *categoría* = ‘Juguetes’; Fuente se lo notifica a Almacén.
2. Almacén pregunta a Fuente los suministradores del producto con $idp = 5$. (*El único suministrador así en ese instante es s1.*)
3. Se inserta la fila $\langle s2, 5 \rangle$ en Suministradores; Fuente se lo notifica a Almacén.
4. Para decidir si se debe añadir $s2$ a la vista hace falta saber la categoría del producto con $idp = 5$, y Almacén se lo pregunta a Fuente. (*Almacén no ha recibido respuesta a su pregunta anterior.*)
5. Fuente procesa ahora la primera consulta de Almacén, halla dos suministradores para el componente 5 y devuelve esa información a Almacén.
6. Almacén recibe la respuesta a su primera pregunta: los suministradores $s1$ y $s2$ y los añade a la vista, cada uno de ellos con un contador igual a 1.
7. Fuente procesa la segunda consulta de Almacén y responde con la información de que el componente 5 es un juguete.
8. Almacén recibe la respuesta a su segunda pregunta e incrementa en consecuencia el contador del suministrador $s2$ en la vista.
9. Ahora se elimina el producto con $idp = 5$; Fuente se lo notifica a Almacén.
10. Dado que el componente eliminado es un juguete, Almacén disminuye los contadores de las tuplas correspondientes de la vista; $s1$ tiene un contador igual a 0 y se elimina, pero $s2$ tiene un contador igual a 1 y se conserva.

Evidentemente, $s2$ no debería permanecer en la vista una vez eliminado el componente 5. Este ejemplo ilustra las sutilezas añadidas del mantenimiento incremental de vistas en entornos distribuidos, y se trata de un tema que actualmente es objeto de investigación.

Vistas para la ayuda a la toma de decisiones. Los fabricantes de SGBD están mejorando sus principales productos relacionales para que soporten las consultas de ayuda a la toma de decisiones. DB2 de IBM soporta las vistas materializadas con mantenimiento consistente con las transacciones o invocado por el usuario. SQL Server de Microsoft soporta las **vistas de particiones**, que son uniones de (muchas) particiones horizontales de una misma tabla. Están orientadas a entornos de almacenamiento en los que cada partición podría ser, por ejemplo, una actualización mensual. Las consultas sobre vistas de particiones se optimizan de modo que sólo se tenga acceso a las participaciones relevantes. Oracle 9i soporta las vistas materializadas con mantenimiento consistente con las transacciones, invocado por el usuario o programado.

16.10.3 Sincronización de las vistas

La **política de mantenimiento de las vistas** es una decisión sobre el momento en que se deben actualizar las vistas, independientemente de si esa actualización es incremental o no. Las vistas se pueden actualizar con la misma transacción que actualiza las tablas subyacentes. Eso se denomina **mantenimiento inmediato de las vistas**. La transacción de actualización se hace más lenta por el paso de actualización, y el efecto de la actualización aumenta con el número de vistas materializadas que dependen de la tabla actualizada.

También se puede posponer la actualización de la vista. Las actualizaciones se capturan en un registro y se aplican posteriormente a las vistas materializadas. Hay varias **políticas de mantenimiento diferido de las vistas**:

1. **Perezosa.** La vista materializada V se actualiza en el momento en que se evalúa una consulta que utilice V , si V ya no es consistente con sus tablas básicas subyacentes. Este enfoque hace más lentas las consultas en lugar de las actualizaciones, a diferencia del mantenimiento inmediato de vistas.
2. **Periódica.** La vista materializada se actualiza de manera periódica, por ejemplo, una vez al día. Muchos fabricantes están ampliando las características de su réplica asíncrona para que soporten las vistas materializadas. Las vistas materializadas que se actualizan periódicamente también se denominan **instantáneas**.
3. **Forzada.** La vista materializada se actualiza una vez realizado un número determinado de modificaciones en las tablas subyacentes.

En el mantenimiento periódico de vistas y en el forzado se pueden considerar las consultas como instancias de la vista materializada que no son consistentes con el estado actual de las tablas subyacentes. Es decir, las consultas verían un conjunto de filas diferente si se volviera a calcular la definición de la vista. Se trata del precio que se paga por tener actualizaciones y consultas rápidas, y la contraprestación es parecida a la que se logra al utilizar la réplica asíncrona.

16.11 PREGUNTAS DE REPASO

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

- ¿Qué son las aplicaciones *de ayuda a la toma de decisiones*? Analícese la relación entre las *consultas complejas de SQL, OLAP, la minería de datos* y el *almacenamiento de datos*. (**Apartado 16.1**)
- Describase el modelo multidimensional de datos. Explíquese la diferencia entre *medidas y dimensiones* y entre *tablas de hechos y tablas de dimensiones*. ¿Qué son los *esquemas en estrella*? (**Apartados 16.2 y 16.2.1**)
- Las operaciones OLAP frecuentes han recibido nombres especiales: *abstracción, concreción, pivotaje, cortes y corte en cubos*. Describase cada una de esas operaciones e ilústrense mediante ejemplos. (**Apartado 16.3**)
- Describanse las características de SQL:1999 ROLLUP y CUBE y su relación con las operaciones OLAP. (**Apartado 16.3.1**)
- Describase la característica WINDOW de SQL:1999, en especial el enmarcado y la ordenación de las ventanas. ¿Cómo soporta las consultas sobre datos ordenados? Dense ejemplos de consultas que sean difíciles de expresar sin esta característica. (**Apartado 16.4**)
- Entre los nuevos paradigmas de consultas se hallan las *N primeras consultas* y la *agregación en línea*. Justifíquense estos conceptos e ilústrense mediante ejemplos. (**Apartado 16.5**)
- Entre las estructuras de índices que resultan especialmente adecuadas para los sistemas OLAP figuran los *índices de mapas de bits* y los *índices de reunión*. Describanse esas estructuras. ¿Cómo están relacionados los índices de mapas de bits con los árboles B+? (**Apartado 16.6**)
- La información sobre las operaciones cotidianas de una organización se guarda en las *bases de datos operativas*. ¿Por qué se emplean *almacenes de datos* para guardar datos de las bases de datos operativas? ¿Qué problemas surgen en el almacenamiento de datos? Analícese la *extracción, la limpieza, la transformación y la carga de datos*. Analíicense los retos de llevar a cabo de manera eficiente la *actualización* y la *purga* de datos. (**Apartado 16.7**)
- ¿Por qué son importantes las vistas en los entornos de ayuda a la toma de decisiones? ¿Cómo están relacionadas las vistas con los almacenes de datos y con OLAP? Explíquese la técnica de *modificación de consultas* para la respuesta a consultas sobre vistas y analícese el motivo de que no resulte adecuada en los entornos de ayuda a la toma de decisiones. (**Apartado 16.8**)
- ¿Cuáles son los principales aspectos que hay que tener en cuenta a la hora del mantenimiento de las vistas materializadas? Analícese el modo de seleccionar las vistas que se deben materializar y la manera de utilizar las vistas materializadas para responder consultas. (**Apartado 16.9**)

- ¿Cómo se pueden mantener las vistas *de manera incremental*? Analíicense todos los operadores del álgebra relacional y la agregación. (**Apartado 16.10.1**)
- Propóngase un ejemplo para ilustrar las complicaciones añadidas en el mantenimiento incremental de las vistas que introduce la distribución de los datos. (**Apartado 16.10.2**)
- Estúdiese la elección de una *política de mantenimiento* adecuada a la hora de actualizar las vistas. (**Apartado 16.10.3**)

EJERCICIOS

Ejercicio 16.1 Respóndase brevemente a las preguntas siguientes:

1. ¿Cómo se complementan entre sí los almacenes de datos, OLAP y la minería de datos?
2. ¿Cuál es la relación entre los almacenes de datos y la réplica de datos? ¿Qué forma de réplica (síncrona o asíncrona) se adapta mejor a los almacenes de datos? ¿Por qué?
3. ¿Cuál es el papel de los repositorios de metadatos en los almacenes de datos? ¿En qué se diferencian de los catálogos de los SGBD relacionales?
4. ¿Qué consideraciones afectan al diseño de los almacenes de datos?
5. Una vez diseñado y cargado un almacén de datos, ¿cómo se mantiene al día con respecto a las modificaciones de las bases de datos fuente?
6. Una de las ventajas de los almacenes es que se pueden utilizar para seguir el modo en que el contenido de una relación cambia con el tiempo; por el contrario, en un SGBD normal sólo se dispone de la instantánea actual. Analícese el modo en que se podría conservar el historial de la relación R , teniendo en cuenta que se debe purgar de algún modo la información “antigua” para dejar espacio a nueva información.
7. Describanse las dimensiones y medidas del modelo multidimensional de datos.
8. ¿Qué son las tablas de hechos y por qué son tan importantes desde el punto de vista del rendimiento?
9. ¿Cuál es la diferencia fundamental entre los sistemas MOLAP y ROLAP?
10. ¿Qué es un esquema en estrella? ¿Se suele hallar en FNBC? ¿Por qué o por qué no?
11. ¿En qué se diferencia la minería de datos de OLAP?

Ejercicio 16.2 Considérese la instancia de la relación Ventas de la Figura 16.2.

1. Muéstrese el resultado de crear tablas dinámicas para idp e $idhora$.
2. Escríbase un conjunto de consultas de SQL para obtener el mismo resultado que en la parte anterior.
3. Muéstrese el resultado de la creación de tablas dinámicas para idp e $idubi$.

Ejercicio 16.3 Considérese la tabla de doble entrada de la relación Ventas mostrada en la Figura 16.5.

1. Muéstrese el resultado de un enrollado sobre $idubi$ (es decir, provincia).
2. Escríbase un conjunto de consultas de SQL para obtener el mismo resultado que en la parte anterior.
3. Muéstrese el resultado de una abstracción sobre $idubi$ seguido de una concreción sobre idp .
4. Escríbase un conjunto de consultas de SQL para obtener el mismo resultado que en la parte anterior, comenzando con la tabla de doble entrada de la Figura 16.5.

Ejercicio 16.4 Respóndase brevemente a las siguientes preguntas:

1. ¿Cuáles son las diferencias entre la cláusula **WINDOW** y la cláusula **GROUP BY**?
2. Dese un ejemplo de consulta que no se pueda expresar en SQL sin la cláusula **WINDOW** pero que sí se pueda expresar con la cláusula **WINDOW**.

3. ¿Qué es el *marco* de una ventana en SQL:1999?
4. Considérese la siguiente consulta sencilla GROUP BY.

```
SELECT    H.año, SUM (V.ventas)
FROM      Ventas V, Horas H
WHERE     V.idhora=H.idhora
GROUP BY  H.año
```

¿Se puede escribir esta consulta en SQL:1999 sin utilizar ninguna cláusula GROUP BY? (Sugerencia: empléese la cláusula WINDOW de SQL:1999.)

Ejercicio 16.5 Considérense las relaciones Ubicaciones, Productos y Ventas de la Figura 16.2. Escribanse las siguientes consultas en SQL:1999 empleando la cláusula WINDOW siempre que sea necesaria.

1. Hallar la variación porcentual de las ventas mensuales totales de cada ubicación.
2. Hallar la variación porcentual de las ventas trimestrales totales de cada producto.
3. Hallar las ventas diarias promedio de los 30 últimos días para cada producto.
4. Para cada semana, hallar la media móvil máxima de ventas durante las cuatro semanas anteriores.
5. Hallar las tres primeras ubicaciones ordenadas por ventas totales.
6. Hallar las tres primeras ubicaciones ordenadas por ventas acumuladas para cada mes del año pasado.
7. Ordenar todas las ubicaciones por ventas totales a lo largo del año pasado e imprimir para cada ubicación la diferencia en ventas totales en relación con la ubicación que la sigue en la clasificación.

Ejercicio 16.6 Considérense la relación Clientes y los índices de mapas de bits de la Figura 16.9.

1. Para los mismos datos, si se supone que el conjunto subyacente de valores de categoría varía de 1 a 10, muéstrese el modo en que cambiarían los índices de mapas de bits.
2. ¿Cómo se utilizarían los índices de mapas de bits para responder a las consultas siguientes? Si los índices de mapas de bits no resultan útiles, explíquese el motivo.
 - (a) ¿Cuántos clientes de categoría menor que 3 son varones?
 - (b) ¿Qué porcentaje de clientes son varones?
 - (c) ¿Cuántos clientes hay?
 - (d) ¿Cuántos clientes se llaman Guillermo?
 - (e) Determíñese la categoría con mayor número de clientes y el número de clientes con esa categoría; si varias categorías tienen el número máximo de clientes, muéstrese la información solicitada para todas ellas. (Supóngase que muy pocas categorías tienen el mismo número de clientes.)

Ejercicio 16.7 Además de la tabla Clientes de la Figura 16.9 con los mapas de bits para *sexo* y *categoría*, supóngase que se tiene una tabla denominada Perspectivas, con los campos *categoría* e *idperspectiva*. Esta tabla se utiliza para identificar posibles clientes.

1. Supóngase que también se tiene un índice de mapa de bits para el campo *categoría* de Perspectivas. Analícese si los índices de mapas de bits ayudarían a calcular la reunión de Clientes y Perspectivas sobre *categoría*.
2. Supóngase que *no* se tiene ningún índice de mapa de bits para el campo *categoría* de Perspectivas. Analícese si los índices de mapa de bits para Clientes ayudarían a calcular la reunión de Clientes y Perspectivas sobre *categoría*.
3. Describábase el empleo de los índices de reunión para soportar la reunión de estas dos relaciones con la condición de reunión *idcli=idperspectiva*.

Ejercicio 16.8 Considérense las instancias de las relaciones Ubicaciones, Productos y Ventas de la Figura 16.2.

1. Considérense los índices de reunión básicos descritos en el Apartado 16.6.2. Supóngase que se desea optimizar para las dos clases de consultas siguientes: la Consulta 1 busca las ventas en una ciudad dada y la Consulta 2 busca las ventas en una provincia dada. Muéstrense los índices que se crearían para los ejemplares de la Figura 16.2.
2. Considérense los índices de reunión para mapas de bits descritos en el Apartado 16.6.2. Supóngase que se desea optimizar para las dos clases de consultas siguientes: la Consulta 1 busca las ventas en una ciudad dada y la Consulta 2 busca las ventas en una provincia dada. Muéstrense los índices que se crearían para los ejemplares de la Figura 16.2.
3. Considérense los índices de reunión básicos descritos en el Apartado 16.6.2. Supóngase que se desea optimizar para las dos clases de consultas siguientes: la Consulta 1 busca las ventas en una ciudad dada para un nombre de producto determinado y la Consulta 2 busca las ventas en una provincia dada para una categoría de producto determinada. Muéstrense los índices que se crearían para los ejemplares de la Figura 16.2.
4. Considérense los índices de reunión para mapas de bits descritos en el Apartado 16.6.2. Supóngase que se desea optimizar para las dos clases de consultas siguientes: la Consulta 1 busca las ventas en una ciudad dada de un nombre de producto determinado y la Consulta 2 busca las ventas en una provincia dada de una categoría de productos dada. Muéstrense los índices que se crearían para los ejemplares de la Figura 16.2.

Ejercicio 16.9 Considérese la vista NúmReservas definida como:

```
CREATE VIEW NúmReservas (idm, nombrem, númres)
AS SELECT M.idm, M.nombrem, COUNT (*)
  FROM    Marineros M, Reservas R
 WHERE   M.idm = R.idm
 GROUP BY M.idm, M.nombrem
```

1. ¿Cómo se reescribiría la consulta siguiente, que pretende hallar el máximo número posible de reservas hechas por un mismo marinero, empleando la modificación de consultas?

```
SELECT MAX (N.númres)
  FROM NúmReservas N
```

2. Considérense las alternativas del cálculo a petición y de materialización de vistas para la consulta anterior. Examínense los pros y contras de la materialización.
3. Examínense los pros y contras de la materialización para la consulta siguiente:

```
SELECT N.nombrem, MAX (N.númres)
  FROM NúmReservas N
 GROUP BY N.nombrem
```

Ejercicio 16.10 Considérense las relaciones Ubicaciones, Productos y Ventas de la Figura 16.2.

1. Para decidir si se materializa una vista, ¿qué factores hay que tener en cuenta?
2. Supóngase que se ha definido la siguiente vista materializada:

```
SELECT U.provincia, V.ventas
  FROM Ubicaciones U, Ventas V
 WHERE V.idubi=U.idubi
```

- (a) Describase la información auxiliar que conserva el algoritmo para el mantenimiento integral de las vistas del Apartado 16.10.1 y el modo en que esos datos ayudan a mantener la vista de manera incremental.
- (b) Examínense los pros y contras de la materialización de esta vista.

3. Considérese la vista materializada de la pregunta anterior. Supóngase que las relaciones Ubicaciones y Ventas se guardan en un sitio, pero que la vista se materializa en un segundo lugar. ¿Qué razones puede haber para desear conservar la vista en un segundo sitio? Dese un ejemplo concreto en el que la vista pueda volverse inconsistente.

4. Supóngase que se ha definido la siguiente vista materializada:

```

SELECT H.año, U.provincia, SUM (V.ventas)
FROM Ventas V, Hora H, Ubicaciones U
WHERE V.idhora=H.idhora AND V.idubi=U.idubi
GROUP BY H.año, U.provincia
    
```

- (a) Describábase la información auxiliar que conserva el algoritmo para el mantenimiento incremental de vistas del Apartado 16.10.1 y el modo en que esos datos ayudan a mantener la vista de manera incremental.
- (b) Analíicense los pros y contras de la materialización de esta vista.

NOTAS BIBLIOGRÁFICAS

En [100], que es la fuente de la Figura 16.10, se presenta un buen resumen de los almacenes de datos y de OLAP. [412] ofrece una visión general de OLAP y de la investigación en las bases de datos estadísticas, que muestra el estrecho paralelismo entre los conceptos y la investigación en esas dos áreas. El libro de Kimball [278], uno de los pioneros en almacenes de datos, y el conjunto de trabajos de [42] ofrecen una buena introducción práctica a esta área. El término OLAP lo popularizó el trabajo de Codd [123]. Para ver un análisis reciente del rendimiento de los algoritmos que utilizan mapas de bits y otras estructuras de índices no tradicionales, consúltese [348].

Stonebraker estudia el modo en el que se pueden convertir las vistas en consultas a las tablas subyacentes mediante la modificación de consultas [433]. Hanson compara el rendimiento de la modificación de consultas con el mantenimiento inmediato y diferido de las vistas [239]. Srivastava y Rotem presentan un modelo analítico de los algoritmos para el mantenimiento de las vistas materializadas [429]. Numerosos trabajos examinan el modo en que se pueden mantener de manera incremental las vistas materializadas a medida que se modifican las relaciones subyacentes. La investigación en esta área se ha vuelto muy activa recientemente, en parte debido al interés en los *almacenes de datos*, que se pueden considerar como conjuntos de vistas sobre las relaciones de varias fuentes. Una excelente visión general de la situación actual se puede hallar en [228], que contiene varios trabajos influyentes, junto con material adicional que proporciona contexto y antecedentes. La siguiente lista parcial debe ofrecer indicaciones para lecturas posteriores: [59, 124, 125, 229, 242, 345, 360, 381, 398, 427, 490].

Gray *et al.* introdujeron el operador CUBE [218], y la optimización de las consultas CUBE y el mantenimiento eficiente del resultado de las consultas CUBE se ha abordado en varios trabajos, incluidos [7, 56, 147, 241, 252, 286, 380, 382, 413, 489]. En [99, 105] se presentan algoritmos relacionados para el procesamiento de consultas con valores agregados y agrupamientos. Rao, Badia y Van Gucht abordan la implementación de consultas que implican cuantificadores generalizados como *la mayor parte de* [371]. Srivastava, Tan y Lum describen un método de acceso para el soporte del procesamiento de consultas de agregación [430]. Shanmugasundaram *et al.* estudian el modo de mantener los cubos comprimidos para la respuesta aproximada de las consultas de agregación en [404].

El soporte de OLAP en SQL:1999, incluidas las estructuras CUBE y WINDOW, se describe en [325]. Las extensiones de ventana son muy parecidas a la ampliación de SQL para la consulta de los datos de secuencias, denominada SRQL, propuesta en [367]. Las consultas de secuencias han recibido mucha atención recientemente. La ampliación de los sistemas relacionales, que trabajan con conjuntos de registros, para que trabajen con secuencias de registros se investiga en [296, 399, 401].

Recientemente ha habido interés en los algoritmos de evaluación de consultas en un solo paso y en la gestión de bases de datos para corrientes de datos. Se puede encontrar un resumen reciente de la gestión de datos par las corrientes de datos y de los algoritmos para el procesamiento de corrientes de datos en [33].

Entre los ejemplos figuran el cálculo de cuantiles y de estadísticas de orden [221, 312], estimación de momentos de frecuencia y de tamaños de reuniones [21, 20], estimación de agregados correlacionados [203], análisis de regresión multidimensional [108] y cálculo de histogramas unidimensionales (es decir, con un solo atributo) y de descomposiciones en ondas de Haar [225, 210].

Otro trabajo propone técnicas para el mantenimiento incremental de histogramas de profundidad constante [206] y de ondas de Haar [320], mantenimiento de muestras y de estadísticas simples en ventanas deslizantes [133], así como arquitecturas generales de alto nivel para los sistemas de bases de datos de corrientes [34]. Zdonik *et al.* describen la arquitectura de un sistema de bases de datos para el control de corrientes de datos [486]. Cortes *et al.* [131] describen la infraestructura de un lenguaje para el desarrollo de aplicaciones de corrientes de datos.

Carey y Kossmann examinan el modo de evaluar las consultas de las que sólo se desean las primeras respuestas [82, 83]. Donjerkovic y Ramakrishnan consideran la manera en que se puede aplicar a este problema un enfoque probabilístico de la optimización de consultas [153]. [70] compara varias estrategias para la evaluación de las primeras N consultas. Hellerstein *et al.* estudian el modo de devolver respuestas aproximadas a las consultas de agregación y de refinrarlas “en línea”. [247, 31]. Este trabajo se ha ampliado al cálculo en línea de reuniones [231], la reordenación en línea [370] y el procesamiento adaptativo de consultas [32].

Ha habido un reciente interés en la respuesta aproximada a las consultas, en la que se utiliza una pequeña estructura de datos de sinopsis para dar respuestas aproximadas rápidas a consultas con garantías de rendimiento que se puedan probar [41, 207, 6, 5, 98, 106, 462].



17

MINERÍA DE DATOS

- ¿Qué es la minería de datos?
- ¿Qué es el análisis del carro de la compra? ¿Qué algoritmos son eficientes para el recuento de las apariciones conjuntas?
- ¿Qué es la propiedad *a priori* y por qué es importante?
- ¿Qué es una red bayesiana?
- ¿Qué es una regla de clasificación? ¿Qué es una regla de regresión?
- ¿Qué es un árbol de decisión? ¿Cómo se crean?
- ¿Qué es la agrupación? ¿Qué es un algoritmo de agrupación de muestras?
- ¿Qué es una búsqueda de similitudes en las secuencias? ¿Cómo se implementa?
- ¿Cómo se pueden crear de manera incremental los modelos de minería de datos?
- ¿Cuáles son los nuevos retos de minería planteados por las corrientes de datos?
- **Conceptos fundamentales:** minería de datos, proceso KDD; análisis de carros de la compra, recuento de apariciones conjuntas, regla de asociación, regla de asociación generalizada; árbol de decisión, árbol de clasificación; agrupación; búsqueda de similitudes de secuencias; mantenimiento incremental del modelo, corrientes de datos, evolución de los bloques.

El secreto del éxito es conocer algo que nadie más conoce.

— Aristóteles Onassis

La **minería de datos** consiste en hallar tendencias o pautas interesantes en conjuntos de datos de gran tamaño para orientar las decisiones sobre actividades futuras. Hay una esperanza generalizada de que las herramientas de minería de datos puedan identificar esas pautas

de los datos con un aporte humano mínimo. Las pautas identificadas por esas herramientas pueden ofrecer a los analistas de datos una perspectiva útil e inesperada que posteriormente se puede investigar con más detenimiento, quizás empleando otras herramientas de soporte a las decisiones. En este capítulo se analizan varias tareas de minería de datos muy estudiadas. Hay herramientas comerciales de los principales fabricantes disponibles para cada una de esas tareas, y el área está creciendo rápidamente en importancia a medida que esas herramientas van obteniendo aceptación entre la comunidad de usuarios.

Se comienza en el Apartado 17.1 ofreciendo una breve introducción a la minería de datos. En el Apartado 17.2 se analiza la importante tarea del recuento de elementos que aparecen conjuntamente. En el Apartado 17.3 se estudia el modo en que surge esta tarea en los algoritmos de minería de datos que descubren reglas a partir de los datos. En el Apartado 17.4 se analizan las pautas que representan reglas en forma de árbol. En el Apartado 17.5 se presenta una tarea de minería de datos diferente, denominada *agrupación*, y se describe la manera de hallar agrupaciones en conjuntos de datos de gran tamaño. En el Apartado 17.6 se describe el modo de llevar a cabo búsquedas de similitudes en las secuencias. En el Apartado 17.7 se estudian los desafíos de la minería en relación con los datos que evolucionan y las corrientes de datos. Se concluye con una breve visión general de otras tareas de minería de datos en el Apartado 17.8.

17.1 INTRODUCCIÓN A LA MINERÍA DE DATOS

La minería de datos está relacionada con la subárea de la estadística denominada *análisis explorador de datos*, que tiene objetivos parecidos y se basa en las medidas estadísticas. También está estrechamente relacionada con las subáreas de la inteligencia artificial denominadas *descubrimiento del conocimiento y aprendizaje de la máquina*. La característica distintiva importante de la minería de datos es que el volumen de datos es muy grande; aunque las ideas de esas áreas de estudio relacionadas sean aplicables a los problemas de minería de datos, la *escalabilidad con respecto al tamaño de los datos* es un criterio nuevo importante. Un algoritmo es **escalable** si el tiempo de ejecución crece (linealmente) en proporción al tamaño del conjunto de datos, lo que mantiene los recursos disponibles del sistema (por ejemplo, la cantidad de memoria principal y la velocidad de proceso de la UCP) constantes. Se deben adaptar los algoritmos antiguos o desarrollar otros nuevos para que se garantice la escalabilidad a la hora de descubrir pautas en los datos.

La búsqueda de tendencias útiles en los conjuntos de datos es una definición bastante imprecisa de la minería de datos: en cierto sentido se puede considerar que todas las consultas a bases de datos hacen exactamente eso. En realidad, hay un continuo de las herramientas de análisis y exploración, con las consultas de SQL en un extremo y las técnicas de minería de datos en el otro. Las consultas de SQL se crean mediante el álgebra relacional (con algunas extensiones), OLAP ofrece expresiones de consulta de nivel superior basadas en el modelo de datos multidimensional, y la minería de datos proporciona las operaciones de análisis más abstractas. Se puede pensar en las diferentes tareas de la minería de datos como “consultas” complejas especificadas en un nivel elevado, con unos cuantos parámetros que son definibles por los usuarios, y para las que se implementan algoritmos especializados.

SQL/MM: Minería de datos SQL/MM. La extensión SQL/MM: Data Mining de la norma SQL:1999 soporta cuatro tipos de modelos de minería de datos: *conjuntos de elementos frecuentes y reglas de asociación, agrupaciones de registros, árboles de regresión y árboles de clasificación*. Se introducen varios tipos de datos nuevos. Estos tipos de datos desempeñan varios papeles. Algunos representan una clase de modelo concreta (por ejemplo, DM_RegressionModel, DM_ClusteringModel); otros especifican los parámetros de entrada de un algoritmo de minería (por ejemplo, DM_RegTask, DM_ClusTask); unos describen los datos de entrada (por ejemplo, DM_LogicalDataSpec, DM_MiningData); y otros representan el resultado de la ejecución de un algoritmo de minería (por ejemplo, DM_RegResult, DM_ClusResult). En conjunto, estas clases y sus métodos ofrecen una interfaz normalizada para los algoritmos de minería de datos que se pueden invocar desde cualquier sistema de bases de datos de SQL:1999. Los modelos de minería de datos se pueden exportar en un formato XML normalizado denominado **Lenguaje de marcas de modelos predictivos** (Predictive Model Markup Language, PMML); también se pueden importar los modelos representados mediante PMML.

En el mundo real la minería de datos es mucho más que la mera aplicación de uno de estos algoritmos. Los datos suelen tener ruido o estar incompletos y, a menos que esto se comprenda y corrija, es probable que muchas pautas interesantes se pasen por alto y la fiabilidad de las detectadas sea baja. Además, el analista debe decidir los tipos de algoritmos de minería que se invocan, aplicarlos a un conjunto bien escogido de muestras de datos y de variables (es decir, tuplas y atributos), resumir los resultados, aplicar otras herramientas de ayuda a la toma decisiones y de minería e iterar el proceso.

17.1.1 El proceso del descubrimiento de conocimiento

El **proceso del descubrimiento de conocimiento y minería de datos** (knowledge discovery and data mining, KDD) puede dividirse grosso modo en cuatro etapas.

1. **Selección de los datos.** El subconjunto de datos objetivo y los atributos de interés se identifican examinando todo el conjunto de datos sin ninguna manipulación previa.
2. **Limpieza de los datos.** Se eliminan el ruido y los datos fuera de rango, se transforman los valores de los campos a unidades comunes y se crean campos nuevos combinando campos ya existentes para facilitar el análisis. Los datos se ponen normalmente en un formato relacional y puede que se combinen varias tablas en una etapa de *desnormalización*.
3. **Minería de datos.** Se aplican los algoritmos de minería de datos para extraer las pautas interesantes.
4. **Evaluación.** Se presentan las pautas a los usuarios finales de manera comprensible, por ejemplo, mediante su visualización.

Puede que el resultado de cualquier etapa del proceso KDD devuelva a una etapa anterior para rehacer el proceso con los nuevos conocimientos obtenidos. En este capítulo, sin embargo,

nos limitaremos a examinar los algoritmos para algunas tareas concretas de minería de datos. No se examinarán otros aspectos del proceso KDD.

17.2 RECUENTO DE APARICIONES CONJUNTAS

Considérese el problema del recuento de los artículos que aparecen conjuntamente, lo cual viene motivado por problemas como el análisis de los carros de la compra. Cada **carro de la compra** es un conjunto de artículos adquiridos por un cliente en una sola **transacción de cliente**. Cada transacción de cliente consiste en una sola visita a la tienda, un solo pedido mediante un catálogo de venta por correo o un pedido a una tienda en Web. (En este capítulo se abreviará a menudo *transacción de cliente* a *transacción* cuando no haya confusión posible con el significado habitual de *transacción* en el contexto de los SGBD, que es la ejecución de un programa de usuario.) Un objetivo frecuente de los comerciantes minoristas es la identificación de los artículos que se compran conjuntamente. Esta información puede emplearse para mejorar la disposición de las mercancías en una tienda o el formato de las páginas de los catálogos.

<i>idtrans</i>	<i>idcli</i>	<i>fecha</i>	<i>artículo</i>	<i>cantidad</i>
111	201	5/1/99	pluma	2
111	201	5/1/99	tinta	1
111	201	5/1/99	leche	3
111	201	5/1/99	zumo	6
112	105	6/3/99	pluma	1
112	105	6/3/99	tinta	1
112	105	6/3/99	leche	1
113	106	5/10/99	pluma	1
113	106	5/10/99	leche	1
114	201	6/1/99	pluma	2
114	201	6/1/99	tinta	2
114	201	6/1/99	zumo	4
114	201	6/1/99	agua	1

Figura 17.1 La relación Compras

17.2.1 Conjuntos de artículos frecuentes

Se usará la relación Compras de la Figura 17.1 para ilustrar los conjuntos de elementos frecuentes. Los registros se muestran ordenados en grupos por transacciones. Todas las tuplas de cada grupo tienen el mismo *idtrans*, y conjuntamente describen la transacción de ese cliente, que supone compras de uno o varios artículos. Cada transacción se produce en una fecha dada y se registra el nombre de cada artículo adquirido, junto con la cantidad comprada. Obsérvese que hay redundancia en Compras: se puede descomponer guardando tripletas *idtrans-idcli-artículo*.

fecha en una tabla diferente y descartando *idcli* y *fecha* de Compras; ésta puede ser la manera en que se guarden realmente los datos. Sin embargo, resulta cómodo considerar la relación Compras, tal y como puede verse en la Figura 17.1, para calcular los conjuntos de artículos frecuentes. La creación de esas tablas “desnormalizadas” para facilitar la minería de datos se suele llevar a cabo en la etapa de limpieza de datos del proceso KDD.

Mediante el examen del conjunto de los grupos de transacciones de Compras se pueden hacer observaciones de la forma: “En el 75% de las transacciones se compran simultáneamente una pluma y tinta.” Esta afirmación describe las transacciones de la base de datos. La extrapolación a transacciones futuras se debe hacer con cautela, como se analiza en el Apartado 17.3.6. Comencemos por presentar la terminología del análisis de cestas de la compra. Un **lote** es un conjunto de artículos. El **soporte** de cada lote es la fracción de las transacciones de la base de datos que contiene todos los artículos de ese conjunto de artículos. En este ejemplo, el lote {pluma, tinta} tiene un 75% de soporte en Compras. Por tanto, se puede concluir que con frecuencia se compran plumas y tinta simultáneamente. Si se considera el lote {leche, zumo}, su soporte es sólo del 25%; la leche y el zumo no se compran simultáneamente con frecuencia.

Normalmente, el número de conjuntos de artículos que se compran simultáneamente con frecuencia es relativamente pequeño, especialmente a medida que aumenta el tamaño de los lotes. Estamos interesados en todos los lotes cuyo soporte es superior a un soporte mínimo especificado por los usuarios, denominado *sopmín*; esos lotes se denominan **lotes frecuentes**. Por ejemplo, si el soporte mínimo se establece en el 70%, entre los lotes frecuentes de este ejemplo están {pluma}, {tinta}, {leche}, {pluma, tinta} y {pluma, leche}. Obsérvese que también estamos interesados en los lotes que sólo contienen un artículo, ya que identifican a los artículos que se adquieren frecuentemente.

En la Figura 17.2 se puede ver un algoritmo para la identificación de lotes frecuentes. Este algoritmo se basa en una propiedad sencilla pero fundamental de los lotes frecuentes:

La propiedad *a priori*. Todo subconjunto de un lote frecuente es también un lote frecuente.

El algoritmo actúa de manera iterativa, identificando primero los lotes frecuentes con un solo artículo. En cada iteración posterior se amplían los lotes frecuentes identificados en la iteración anterior con otro artículo para generar posibles lotes de mayor tamaño. Al considerar únicamente los lotes obtenidos al ampliar los lotes frecuentes, se reduce enormemente el número de lotes frecuentes posibles; esta optimización resulta crucial para una ejecución eficiente. La propiedad *a priori* garantiza que esta optimización es correcta; es decir, no se pasa por alto ningún lote frecuente. Basta un único examen de todas las transacciones (la relación Compras de este ejemplo) para determinar cuáles de los posibles lotes generados en cada iteración son frecuentes. El algoritmo concluye cuando en alguna iteración no se identifica ningún lote frecuente nuevo.

El algoritmo se ilustra para la relación Compras en la Figura 17.1, con *sopmín* establecido en el 70%. En la primera iteración (Nivel 1) se examina la relación Compras y se determina que todos los conjuntos de un solo artículo son lotes frecuentes: {pluma} (aparece en las cuatro transacciones), {tinta} (aparece en tres de las cuatro transacciones) y {leche} (aparece en tres de las cuatro transacciones).

```

foreach item,                                     // Nivel 1
    comprobar si se trata de un lote frecuente // aparece en > sopmín transacciones
k = 1
repeat                                         // Identificación iterativa por niveles de lotes frecuentes
    para cada lote frecuente nuevo  $L_k$  con  $k$  artículos          // Nivel  $k + 1$ 
        generar todos los lotes  $L_{k+1}$  con  $k + 1$  artículos,  $L_k \subset L_{k+1}$ 
        Examinar todas las transacciones una vez y comprobar si
        los  $k + 1$  lotes generados son frecuentes
         $k = k + 1$ 
until no se identifica ningún lote frecuente nuevo

```

Figura 17.2 Algoritmo para la búsqueda de lotes frecuentes

En la segunda iteración (Nivel 2) se amplían todos los lotes frecuentes con un artículo adicional y se generan los siguientes lotes posibles: $\{pluma, tinta\}$, $\{pluma, leche\}$, $\{pluma, zumo\}$, $\{tinta, leche\}$, $\{tinta, zumo\}$ y $\{leche, zumo\}$. Al examinar nuevamente la relación Compras se determina que los lotes siguientes son frecuentes: $\{pluma, tinta\}$ (aparece en tres de las cuatro transacciones) y $\{pluma, leche\}$ (aparece en tres de las cuatro transacciones).

En la tercera iteración (Nivel 3) se amplían estos lotes con un artículo adicional y se generan los siguientes lotes posibles: $\{pluma, tinta, leche\}$, $\{pluma, tinta, zumo\}$ y $\{pluma, leche, zumo\}$. (Obsérvese que no se genera $\{tinta, leche, zumo\}$.) Un tercer examen de la relación Compras permite determinar que ninguno de estos lotes es frecuente.

El algoritmo sencillo para la búsqueda de lotes frecuentes aquí presentado ilustra la característica principal de los algoritmos más sofisticados, es decir, la generación y el examen iterativos de los posibles lotes. A continuación se considerará una mejora importante de este algoritmo sencillo. La creación de posibles lotes mediante la inserción de un elemento a lotes frecuentes ya conocidos es un intento de limitar el número de lotes posibles mediante el empleo de la propiedad *a priori*. La propiedad *a priori* implica que cada lote posible sólo puede ser frecuente si todos sus subconjuntos lo son. Por tanto, se puede reducir aún más el número de lotes posibles —*a priori*, o antes de examinar la base de datos Compras— comprobando si todos los subconjuntos de los posibles lotes recién generados son frecuentes. Sólo se calcula su soporte en el examen posterior de la base de datos si todos los subconjuntos de un posible lote son frecuentes. Comparado con el algoritmo sencillo, este algoritmo refinado genera menos lotes posibles en cada nivel y, por tanto, reduce la cantidad de cálculo llevada a cabo durante el examen de la base de datos de Compras.

Considérese el algoritmo refinado de la tabla Compras de la Figura 17.1 con $sopmín= 70\%$. En la primera iteración (Nivel 1) se determinan los lotes frecuentes de tamaño uno: $\{pluma\}$, $\{tinta\}$ y $\{leche\}$. En la segunda iteración (Nivel 2) sólo quedan los siguientes lotes posibles al examinar la tabla Compras: $\{pluma, tinta\}$, $\{pluma, leche\}$ y $\{tinta, leche\}$. Dado que $\{zumo\}$ no es frecuente, los lotes $\{pluma, zumo\}$, $\{tinta, zumo\}$ y $\{leche, zumo\}$ no pueden ser también frecuentes y se pueden eliminar *a priori*, es decir, sin considerarlos durante el examen posterior de la relación Compras. En la tercera iteración (Nivel 3) ya no se generan más lotes posibles. El lote $\{pluma, tinta, leche\}$ no puede ser frecuente, ya que su subconjunto

$\{tinta, leche\}$ no lo es. Por tanto, la versión mejorada del algoritmo no necesita un tercer examen de Compras.

17.2.2 Consultas iceberg

Las consultas iceberg se introducirán mediante un ejemplo. Considérese nuevamente la relación Compras de la Figura 17.1. Supóngase que se desea hallar parejas de clientes y artículos tales que el consumidor haya comprado ese artículo más de cinco veces. Esta consulta se puede expresar en SQL de la manera siguiente:

```
SELECT C.idcli, C.producto, SUM (C.cantidad)
FROM Compras C
GROUP BY C.idcli, C.producto
HAVING SUM (C.cantidad) > 5
```

Imagínese cómo evaluaría esta consulta un SGBD relacional. Conceptualmente, para cada pareja ($idcli, producto$), hay que comprobar si la suma del campo *cantidad* es mayor de 5. Un enfoque es examinar la relación Compras y mantener sumas por cada pareja ($idcli, producto$). Se trata de una estrategia de ejecución factible siempre que el número de parejas sea lo bastante pequeño como para caber en la memoria principal. Si el número de parejas es mayor que la memoria principal hay que utilizar planes de ejecución de consultas más costosos, que comprendan ordenaciones o asociaciones.

La consulta tiene una propiedad importante no aprovechada por la estrategia de ejecución anterior: aunque la relación Compras pueda llegar a ser muy grande y el número de grupos ($idcli, producto$) enorme, es probable que el resultado de la consulta sea relativamente pequeño debido a la condición de la cláusula HAVING. Sólo aparecen en el resultado los grupos en los que el cliente ha comprado ese artículo más de cinco veces. Por ejemplo, en la consulta a la relación Compras de la Figura 17.1 hay nueve grupos, aunque el resultado sólo contiene tres registros. El número de grupos es muy grande, pero la respuesta a la consulta —la punta del iceberg— suele ser muy pequeña. Por tanto, esas consultas se denominan **consultas iceberg**. En general, dado un esquema relacional R con los atributos A_1, A_2, \dots, A_k y B y la función de agregación **agre**, la consulta iceberg tiene la estructura siguiente:

```
SELECT R.A1, R.A2, ..., R.Ak, agre(R.B)
FROM Relación R
GROUP BY R.A1, ..., R.Ak
HAVING agre(R.B) >= constante
```

Los planes de ejecución tradicionales para esta consulta que emplean la ordenación o la asociación calculan primero el valor de la función de agregación para todos los grupos y luego eliminan los grupos que no satisfacen la condición de la cláusula HAVING.

Al comparar esta consulta con el problema de la búsqueda de lotes frecuentes estudiado en el apartado anterior surge una similitud sorprendente. Considérense nuevamente la relación Compras de la Figura 17.1 y la consulta iceberg del comienzo de este apartado. Estamos interesados en las parejas ($idcli, producto$) que tienen $\text{SUM} (C.cantidad) > 5$. Si empleamos una variación de la propiedad *a priori* se puede aducir que sólo hay que tomar en consideración

los valores del campo *idcli* en los que el cliente ha comprado, como mínimo, cinco artículos. Esos artículos se pueden obtener mediante la consulta siguiente:

```
SELECT    C.idcli
FROM      Compras C
GROUP BY  C.idcli
HAVING   SUM (C.cantidad) > 5
```

De manera análoga, se pueden reducir los posibles valores del campo *producto* mediante la consulta siguiente:

```
SELECT    C.producto
FROM      Compras C
GROUP BY  C.producto
HAVING   SUM (C.cantidad) > 5
```

Si se restringe el cálculo de la consulta iceberg original a grupos (*idcli*, *producto*) en los que los valores de los campos sean el resultado de las dos consultas anteriores, se elimina *a priori* gran número de parejas (*idcli*, *producto*). Por tanto, una estrategia de evaluación posible es calcular primero los valores posibles de los campos *idcli* y *producto* y sólo emplear combinaciones de esos valores en la evaluación de la consulta iceberg original. En primer lugar se obtienen los valores posibles de los campos para cada uno de esos campos y sólo se utilizan los valores que sobreviven el paso de selección *a priori*, tal y como se expresa en las dos consultas anteriores. Así, la consulta iceberg resulta asequible a la misma estrategia de evaluación ascendente empleada para hallar los lotes frecuentes. En concreto, se puede emplear la propiedad *a priori* de la manera siguiente: sólo se utiliza un contador para un grupo dado si cada uno de los componentes de ese grupo satisface la condición expresada en la cláusula HAVING. Las mejoras de rendimiento de esta estrategia de evaluación alternativa respecto de los planes de ejecución de la consulta tradicionales pueden resultar muy significativas en la práctica.

Aunque la estrategia de procesamiento de consultas ascendente elimine *a priori* muchos grupos, en la práctica el número de parejas (*idcli*, *producto*) puede seguir siendo muy grande; más grande incluso que la memoria principal. Se han desarrollado estrategias eficientes que emplean el muestreo y técnicas de asociación más sofisticadas; las notas bibliográficas al final del capítulo ofrecen indicaciones de la literatura relevante para el caso.

17.3 MINERÍA DE REGLAS

Se han propuesto muchos algoritmos para el descubrimiento de diferentes formas de reglas que describan sucintamente los datos. A continuación se examinarán algunas de estas reglas y algoritmos para descubrirlas que han sido muy estudiados.

17.3.1 Reglas de asociación

Se usará la relación Compras de la Figura 17.1 para ilustrar las reglas de asociación. Mediante el examen del conjunto de transacciones de Compras se pueden identificar reglas de la forma:

$$\{pluma\} \Rightarrow \{tinta\}$$

Esta regla se debe leer de la manera siguiente: “Si en una transacción se compra una pluma, es probable que también se compre tinta en esa transacción.” Es una afirmación que describe las transacciones de la base de datos; la extrapolación a transacciones futuras debe hacerse con cautela, como se analiza en el Apartado 17.3.6. Más en general, las **reglas de asociación** tienen la forma $Izq \Rightarrow Der$, donde tanto Izq como Der son conjuntos de elementos. La interpretación de esta regla es que si se compran en una transacción todos los artículos de Izq , entonces es probable que también se compren los artículos de Der .

Hay dos medidas importantes para las reglas de asociación:

- **Soporte.** El soporte de un conjunto de artículos es el porcentaje de transacciones que contienen todos esos artículos. El soporte de la regla $Izq \Rightarrow Der$ es el soporte del conjunto de artículos $Izq \cup Der$. Por ejemplo, considérese la regla $\{pluma\} \Rightarrow \{tinta\}$. El soporte de esta regla es el soporte del lote $\{pluma, tinta\}$, que es 75%.
- **Confianza.** Considérense transacciones que contengan todos los artículos de Izq . La confianza de la regla $Izq \Rightarrow Der$ es el porcentaje de esas transacciones que contienen también todos los artículos de Der . Más exactamente, sea $sop(Izq)$ el porcentaje de transacciones que contienen Izq y $sop(Izq \cup Der)$ el porcentaje de transacciones que contienen tanto Izq como Der . Entonces, la confianza de la regla $Izq \Rightarrow Der$ es $sop(Izq \cup Der) / sop(Izq)$. La confianza de cada regla es una indicación de su fortaleza. Por ejemplo, considérese nuevamente la regla $\{pluma\} \Rightarrow \{tinta\}$. La confianza de esta regla es 75%; el 75% de las transacciones que contienen el lote $\{pluma\}$ contienen también el lote $\{tinta\}$.

17.3.2 Algoritmo para la búsqueda de reglas de asociación

Un usuario puede pedir todas las reglas de asociación que tengan un soporte mínimo determinado ($sopmín$) y una confianza mínima ($confmín$), y se han desarrollado varios algoritmos para hallar esas reglas de manera eficiente. Esos algoritmos trabajan en dos etapas. En la primera etapa se calculan todos los lotes frecuentes con el soporte mínimo especificado por el usuario. En la segunda etapa se generan reglas empleando como datos los lotes frecuentes. En el Apartado 17.2 se analizó un algoritmo para la búsqueda de lotes frecuentes; aquí nos centraremos en lo relativo a la generación de reglas.

Una vez identificados los lotes frecuentes, la generación de todas las reglas posibles con el soporte mínimo especificado por el usuario es sencilla. Considérese el lote frecuente X con el soporte s_X identificado en la primera etapa del algoritmo. Para generar una regla a partir de X se divide X en dos lotes, Izq y Der . La confianza de la regla $Izq \Rightarrow Der$ es s_X / s_{Izq} , la relación entre el soporte de X y el de Izq . A partir de la propiedad *a priori* se sabe que el soporte de Izq es mayor que $sopmín$ y, por tanto, se ha calculado el soporte de Izq en la primera etapa del algoritmo. Los valores de confianza de la posible regla se pueden obtener calculando la relación $sop(X) / sop(Izq)$ y, luego, comprobando su relación con $sopmín$.

En general, la etapa más costosa del algoritmo es el cálculo de los lotes frecuentes, y se han desarrollado muchos algoritmos diferentes para llevarla a cabo de manera eficiente. La generación de reglas —dado que ya se han identificado todos los lotes frecuentes— es sencilla.

En el resto de este apartado se analizan algunas generalizaciones del problema.

17.3.3 Reglas de asociación y jerarquías ES

En muchos casos se impone una **jerarquía ES** o **jerarquía categórica** al conjunto de artículos. En presencia de jerarquías cada transacción contiene implícitamente, para cada uno de sus artículos, todos los ancestros de esos artículos en la jerarquía. Por ejemplo, considérese la jerarquía de categorías de la Figura 17.3. Dada esta jerarquía, la relación Compras se incrementa conceptualmente con los ocho registros que aparecen en la Figura 17.4. Es decir, la relación Compras tiene todas las tuplas que se pueden ver en la Figura 17.1 además de las tuplas de la Figura 17.4.

La jerarquía permite detectar las relaciones entre artículos de diferentes niveles de la jerarquía. Por ejemplo, el soporte del lote $\{tinta, zumo\}$ es del 50%, pero si se sustituye *zumo* por la categoría más general *bebidas*, el soporte del lote resultante $\{tinta, bebidas\}$ aumenta al 75%. En general, el soporte de cada lote sólo puede aumentar si se sustituye un artículo por alguno de sus ancestros de la jerarquía ES.

Suponiendo que se añaden físicamente de verdad los ocho registros de la Figura 17.4 a la relación Compras, se puede emplear cualquier algoritmo para el cálculo de los lotes frecuentes de la base de datos incrementada. Suponiendo que la jerarquía quepa en memoria principal, también se puede llevar a cabo la adición sobre la marcha mientras se examina la base de datos, a modo de optimización.



Figura 17.3 Taxonomía de una categoría ES

<i>idtrans</i>	<i>idcli</i>	<i>fecha</i>	<i>producto</i>	<i>cantidad</i>
111	201	5/1/99	papelería	3
111	201	5/1/99	bebidas	9
112	105	6/3/99	papelería	2
112	105	6/3/99	bebidas	1
113	106	5/10/99	papelería	1
113	106	5/10/99	bebidas	1
114	201	6/1/99	papelería	4
114	201	6/1/99	bebidas	5

Figura 17.4 Inserciones en la relación Compras con una jerarquía ES

17.3.4 Reglas de asociación generalizadas

Aunque las reglas de asociación se han estudiado muy a fondo en el contexto del análisis de las cestas de la compra, o análisis de las transacciones de los consumidores, el concepto es más general. Considérese la relación Compras tal y como se puede ver en la Figura 17.5, agrupada por *idcli*. Mediante el examen del conjunto de grupos de clientes se pueden identificar reglas de asociación como $\{\text{pluma}\} \Rightarrow \{\text{leche}\}$. Esta regla se debe leer ahora de la manera siguiente: “Si un cliente compra una pluma, es probable que ese cliente también compre leche.” En la relación Compras de la Figura 17.5 esta regla tiene soporte y confianza del 100%.

<i>idtrans</i>	<i>idcli</i>	<i>fecha</i>	<i>producto</i>	<i>cantidad</i>
112	105	6/3/99	pluma	1
112	105	6/3/99	tinta	1
112	105	6/3/99	leche	1
113	106	5/10/99	pluma	1
113	106	5/10/99	leche	1
114	201	5/15/99	pluma	2
114	201	5/15/99	tinta	2
114	201	5/15/99	zumo	4
114	201	6/1/99	agua	1
111	201	5/1/99	pluma	2
111	201	5/1/99	tinta	1
111	201	5/1/99	leche	3
111	201	5/1/99	zumo	6

Figura 17.5 La relación Compras ordenada por identificador de cliente

De manera parecida, las tuplas se pueden agrupar por fecha e identificar reglas de asociación que describan el comportamiento de compra en un mismo día. Por ejemplo, considérese nuevamente la relación Compras. En este caso, la regla $\{\text{pluma}\} \Rightarrow \{\text{leche}\}$ se interpreta ahora de la manera siguiente: “En el día en que se compra una pluma es probable que también se compre leche.”

Si se utiliza el campo *fecha* como atributo de agrupación, se puede considerar un problema más general denominado **análisis de cestas de la compra de acuerdo con el calendario**. En el análisis de cestas de la compra de acuerdo con el calendario el usuario especifica un conjunto de **calendarios**. Calendario es cualquier grupo de fechas como, por ejemplo, *todos los domingos del año 1999*, o *todos los primeros de mes*. Una regla se cumple si se cumple en todos los días de ese calendario. Dado un calendario, se pueden calcular reglas de asociación para el conjunto de tuplas cuyo campo *fecha* esté incluido en ese calendario.

Mediante la especificación de calendarios interesantes se pueden identificar reglas que quizás no tengan soporte y confianza suficientes con respecto a toda la base de datos pero sí en el subconjunto de tuplas incluidas en esos calendarios. Por otro lado, aunque una regla tenga soporte y confianza suficientes con respecto a toda la base de datos, puede que sólo obtenga su soporte de las tuplas incluidas en un calendario. En ese caso, el soporte de la regla

en las tuplas del calendario es significativamente mayor que su soporte con respecto a toda la base de datos.

Por ejemplo, considérese la relación Compras con el calendario *todos los primeros de mes*. En ese calendario la regla de asociación $pluma \Rightarrow zumo$ tiene soporte y confianza del 100%, mientras que en toda la relación Compras esa regla sólo tiene un soporte del 50%. Por otro lado, la regla $pluma \Rightarrow leche$ tiene en el calendario un soporte y una confianza del 50%, mientras que en toda la relación Compras tiene un soporte y una confianza del 75%.

También se han propuesto especificaciones más generales de las condiciones que deben cumplirse en un grupo para que se cumpla una regla (para ese grupo). Puede que se desee afirmar que los artículos de *Izq* se deben comprar en grupos de menos de dos y que los artículos de *Der* se deben comprar en grupos de más de tres.

Mediante el empleo de diferentes opciones para el atributo de agrupación y de condiciones sofisticadas como en los ejemplos anteriores, se pueden identificar reglas más complejas que las reglas de asociación básicas analizadas anteriormente. No obstante, estas reglas más complejas conservan la estructura básica de las reglas de asociación como condiciones sobre grupos de tablas, con las medidas de soporte y de confianza definidas como siempre.

17.3.5 Pautas secuenciales

Considérese la relación Compras de la Figura 17.1. Cada grupo de tuplas con el mismo valor de *idcli* se puede considerar una *secuencia* de transacciones ordenadas por *fecha*. Esto permite la identificación de pautas de compra que aparecen con frecuencia a lo largo del tiempo.

Comencemos introduciendo el concepto de secuencia de lotes. Cada transacción se representa mediante un conjunto de tuplas y, examinando los valores de la columna *producto*, se obtiene el conjunto de artículos comprados en esa transacción. Por tanto, la secuencia de transacciones asociadas a un cliente se corresponde de manera natural con la secuencia de lotes comprados por ese cliente. Por ejemplo, la secuencia de compras del comprador 201 es $\langle\{pluma, tinta, leche, zumo\}, \{pluma, tinta, zumo\}\rangle$.

Las **subsecuencias** de las secuencias de lotes se obtienen eliminando uno o varios lotes y son, a su vez, secuencias de lotes. Se dice que la secuencia $\langle a_1, \dots, a_m \rangle$ está **contenida** en la secuencia S si S tiene una subsecuencia $\langle b_1, \dots, b_m \rangle$ tal que $a_i \subseteq b_i$, para $1 \leq i \leq m$. Por tanto, la secuencia $\langle\{pluma\}, \{tinta, leche\}, \{pluma, zumo\}\rangle$ está contenida en $\langle\{pluma, tinta\}, \{camisa\}, \{zumo, tinta, leche\}, \{zumo, pluma, leche\}\rangle$. Obsérvese que el orden de los artículos dentro de cada lote no tiene importancia. Sin embargo, el orden de los lotes sí que importa: la secuencia $\langle\{pluma\}, \{tinta, leche\}, \{pluma, zumo\}\rangle$ no está contenida en $\langle\{pluma, tinta\}, \{camisa\}, \{zumo, pluma, leche\}, \{zumo, leche, tinta\}\rangle$.

El **soporte** de la secuencia de lotes S es el porcentaje de secuencias de clientes de las que S es subsecuencia. El problema de la identificación de las pautas secuenciales es hallar todas las secuencias que tienen el soporte mínimo especificado por el usuario. La secuencia $\langle a_1, a_2, a_3, \dots, a_m \rangle$ con soporte mínimo indica que los usuarios compran a menudo los artículos del lote a_1 en una transacción, luego compran en una transacción posterior los artículos del lote a_2 , luego los del lote a_3 en una transacción posterior, etcétera.

Al igual que las reglas de asociación, las pautas secuenciales son afirmaciones sobre grupos de tuplas de la base de datos actual. En términos de cálculo, los algoritmos para buscar pautas

secuenciales que aparezcan con frecuencia recuerdan a los algoritmos para buscar los lotes frecuentes. Se identifican de manera iterativa secuencias cada vez más largas con el soporte mínimo exigido de manera muy parecida a la identificación iterativa de lotes frecuentes.

17.3.6 Empleo de las reglas de asociación para la predicción

Las reglas de asociación se utilizan mucho para la predicción, pero es importante reconocer que ese empleo predictivo no está justificado sin un análisis adicional o cierto conocimiento del dominio. Las reglas de asociación describen con precisión los datos existentes, pero pueden llevar a error si se utilizan de manera ingenua para la predicción. Por ejemplo, considérese la regla

$$\{pluma\} \Rightarrow \{tinta\}$$

La confianza asociada con esta regla es la probabilidad condicional de la compra de tinta dada la compra de una pluma *en la base de datos dada*; es decir, se trata de una medida *descriptiva*. Esta regla se podría utilizar para orientar futuras promociones comerciales. Por ejemplo, se podría ofrecer un descuento en las plumas para aumentar sus ventas y, en consecuencia, también las de tinta.

Sin embargo, una promoción así supone que las compras de plumas son buenos indicadores de las compras de tinta en *futuras* transacciones de los clientes (además de serlo en las transacciones de la base de datos actual). Esta suposición está justificada si existe un *vínculo causal* entre las compras de plumas y las de tinta; es decir, si las compras de plumas hacen que el comprador también compre tinta. Sin embargo, se pueden deducir reglas de asociación con soporte y confianza elevados en algunas situaciones en las que no existe ningún vínculo causal entre el *Izq* y el *Der*. Por ejemplo, supóngase que las plumas se compran siempre junto con lápices, quizás debido a la tendencia de los clientes a pedir juntos los instrumentos de escritura. Se podría deducir la regla

$$\{lápiz\} \Rightarrow \{tinta\}$$

con el mismo soporte y la misma confianza que la regla

$$\{pluma\} \Rightarrow \{tinta\}$$

Sin embargo, no hay vínculo causal alguno entre los lápices y la tinta. Si se promocionan los lápices, el cliente que compre varios lápices debido a la promoción no tiene razón alguna para comprar más tinta. Por tanto, una promoción comercial que ofreciera descuentos en los lápices para aumentar las ventas de tinta fracasaría.

En la práctica, cabe esperar que, mediante el examen de una base de datos de gran tamaño de transacciones pasadas (reunida a lo largo de mucho tiempo y gran variedad de circunstancias) y restringiendo nuestra atención a reglas que se den a menudo (es decir, que tengan gran soporte), se minimice la deducción de reglas que induzcan a error. No obstante, se debe tener presente que se pueden seguir generando reglas no causales que induzcan a error. Por tanto, se deben tratar las reglas generadas como si posiblemente, más que concluyentemente, identificaran relaciones causales. Aunque las reglas de asociación no indican relaciones causales entre *Izq* y *Der*, hay que subrayar que ofrecen un punto de partida útil para la identificación de esas relaciones, bien mediante un análisis más profundo, bien mediante el juicio de expertos en el dominio correspondiente; a ello se debe su popularidad.

17.3.7 Redes bayesianas

La búsqueda de relaciones causales supone un desafío, como se vio en el Apartado 17.3.6. En general, si determinados sucesos están muy correlacionados, hay muchas explicaciones posibles. Por ejemplo, supóngase que las plumas, los lápices y la tinta se compran juntos con frecuencia. Pudiera ser que la compra de uno de estos artículos (por ejemplo, la tinta) dependiera causalmente de la compra de otro (por ejemplo, las plumas). O bien pudiera ser que la compra de uno de esos artículos (por ejemplo, las plumas) estuviese fuertemente correlacionado con la compra de otro de ellos (por ejemplo, los lápices) debido a algún fenómeno subyacente (por ejemplo, la tendencia de los usuarios a pensar en los instrumentos de escritura conjuntamente) que influye causalmente en ambas compras. ¿Cómo se identifican las relaciones causales que se cumplen realmente entre estos sucesos en el mundo real?

Un enfoque es considerar cada posible combinación de relaciones causales entre las variables de los sucesos de interés y evaluar la posibilidad de cada combinación con base en los datos disponibles. Si se considera cada combinación de relaciones causales como un *modelo* del mundo real subyacente a los datos recolectados, se puede asignar una puntuación a cada modelo considerando su consistencia (en términos de probabilidades, con algunas suposiciones simplificadoras) con los datos observados. Las redes bayesianas son grafos que se pueden utilizar para describir una clase de estos modelos, con un nodo por variable o suceso y arcos entre los nodos para indicar la causalidad. Por ejemplo, un buen modelo del ejemplo de plumas, lápices y tinta puede verse en la Figura 17.6. En general, el número de modelos posibles es exponencial en el número de variables, y considerar todos los modelos resulta costoso, por lo que se evalúa algún subconjunto de todos los modelos posibles.

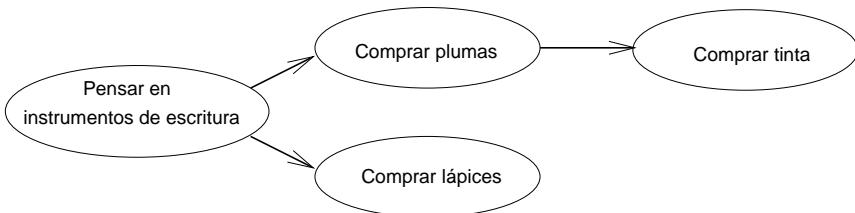


Figura 17.6 Red bayesiana que muestra causalidad

17.3.8 Reglas de clasificación y de regresión

Considérese la vista siguiente, que contiene información de una campaña de publicidad por correo llevada a cabo por una empresa de seguros:

`InfoSeguro(edad: integer, tipocoche: string, riesgoalto: boolean)`

La vista InfoSeguros tiene información sobre los clientes actuales. Cada registro contiene la edad y tipo de coche de un cliente, así como un indicador que indica si esa persona está considerada cliente de alto riesgo. Si el indicador es verdadero, el cliente se considera de alto riesgo. Nos gustaría utilizar esta información para identificar reglas que predigan el riesgo para la aseguradora de los nuevos solicitantes de seguros cuya edad y tipo de coche

sean conocidos. Por ejemplo, una de esas reglas podría ser: “Si *edad* está entre 18 y 25 y *tipocoche* es Deportivo o Camión, el riesgo es elevado.”

Obsérvese que las reglas que se desea hallar tienen una estructura concreta. No estamos interesados en reglas que predigan la edad o el tipo de coche de una persona dada; sólo estamos interesados en las que predicen el riesgo para la aseguradora. Por tanto, hay un atributo concreto cuyo valor se desea predecir, y a ese atributo se le denomina atributo **dependiente**. Los demás atributos se denominan atributos **predictores**. En este ejemplo, el atributo dependiente de la vista InfoSeguro es *riesgoalto* y los atributos predictores son *edad* y *tipocoche*. La forma general del tipo de reglas que se desea descubrir es:

$$P_1(X_1) \wedge P_2(X_2) \dots \wedge P_k(X_k) \Rightarrow Y = c$$

Los atributos predictores X_1, \dots, X_k se utilizan para predecir el valor del atributo dependiente Y . Los dos lados de la regla se pueden interpretar como condiciones para los campos de una tupla. $P_i(X_i)$ son predicados que implican al atributo X_i . La forma de cada predicado depende del tipo del atributo predictor. Se distinguen dos tipos de atributos: numéricos y categóricos. Para los atributos **numéricos** se pueden llevar a cabo cálculos numéricos, como el cálculo del promedio de dos valores; mientras que para los atributos **categóricos** la única operación permitida es comprobar si dos valores son iguales. En la vista InfoSeguros *edad* es un atributo numérico, mientras que *tipocoche* y *riesgoalto* son atributos categóricos. Volviendo a la forma de los predicados, si X_i es un atributo numérico, su predicado P_i es de la forma $inf_i \leq X_i \leq sup_i$; si X_i es un atributo categórico, P_i es de la forma $X_i \in \{v_1, \dots, v_j\}$.

Si el atributo dependiente es categórico, las reglas se denominan **reglas de clasificación**. Si el atributo dependiente es numérico, las reglas se denominan **reglas de regresión**.

Por ejemplo, considérese nuevamente la regla del ejemplo: “Si *edad* está entre 18 y 25 y *tipocoche* es Deportivo o Camión, *riesgoalto* es verdadero.” Dado que *riesgoalto* es un atributo categórico, esta regla es una regla de clasificación. Se puede expresar formalmente de la manera siguiente:

$$(18 \leq edad \leq 25) \wedge (tipocoche \in \{\text{Deportivo, Camión}\}) \Rightarrow riesgoalto = \text{verdadero}$$

Se pueden definir el soporte y la confianza para las reglas de clasificación y de regresión igual que para las reglas de asociación:

- **Soporte.** El soporte de la condición C es el porcentaje de tuplas que satisfacen C . El soporte de la regla $C1 \Rightarrow C2$ es el soporte de la condición $C1 \wedge C2$.
- **Confianza.** Considérense las tuplas que satisfacen la condición $C1$. La confianza de la regla $C1 \Rightarrow C2$ es el porcentaje de esas tuplas que satisfacen también la condición $C2$.

Como generalización adicional, considérese el segundo miembro de una regla de clasificación o de regresión: $Y = c$. Cada regla predice un valor de Y para una tupla dada de acuerdo con el valor de los atributos predictores X_1, \dots, X_k . Se pueden considerar reglas de la forma:

$$P_1(X_1) \wedge \dots \wedge P_k(X_k) \Rightarrow Y = f(X_1, \dots, X_k)$$

donde f es una función. Estas reglas no se estudiarán más a fondo.

Las reglas de clasificación y las de regresión se diferencian de las reglas de asociación en que tienen en consideración campos continuos y categóricos, en lugar de un solo campo que toma un conjunto de valores. La identificación eficiente de esas reglas plantea una nueva serie de retos; el caso general del descubrimiento de esas reglas no se estudiará aquí. Se analizará un tipo especial de este tipo de reglas en el Apartado 17.4.

Las reglas de clasificación y las de regresión tienen muchas aplicaciones. Entre los ejemplos figuran la clasificación de los resultados de los experimentos científicos, en los que el tipo de objeto que se debe reconocer depende de las medidas tomadas; la prospección mediante correo directo, en la que la respuesta de un cliente dado a una promoción concreta es función de su nivel de renta y de su edad; y la evaluación de riesgo para el seguro de automóvil, en la que los clientes se pueden clasificar como de riesgo en función de su edad, profesión y tipo de coche. Entre los ejemplos de aplicaciones de las reglas de regresión están la predicción financiera, en la que el precio de los futuros sobre el café puede ser una función de la lluvia caída en Colombia hace un mes, y el pronóstico en medicina, en el que la probabilidad de que un tumor sea canceroso es función de las medidas de sus atributos.

17.4 REGLAS ESTRUCTURADAS EN ÁRBOLES

En este apartado se analizará el problema del descubrimiento de reglas de clasificación y de regresión a partir de una relación, pero sólo se tomarán en consideración las reglas que tengan una estructura muy especial. El tipo de reglas que se estudiará se puede representar mediante un árbol y, habitualmente, el propio árbol es el resultado de la actividad de minería de datos. Los árboles que representan las reglas de clasificación se denominan **árboles de clasificación** o **árboles de decisión** y los árboles que representan las reglas de regresión se denominan **árboles de regresión**.

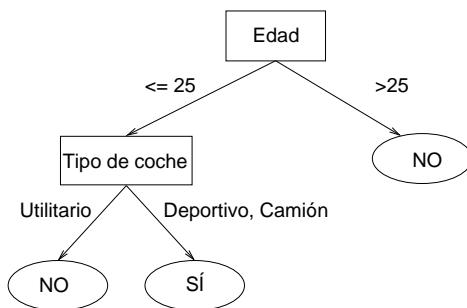


Figura 17.7 Ejemplo de árbol de decisión para el riesgo de un seguro

Por ejemplo, considérese el árbol de decisión de la Figura 17.7. Cada ruta desde el nodo raíz hasta un nodo hoja representa una regla de clasificación. Por ejemplo, la ruta desde la raíz hasta el nodo hoja situado más a la izquierda representa la regla de clasificación: "Si una persona tiene 25 años o menos y conduce un utilitario, es probable que suponga un riesgo bajo para la aseguradora." La ruta desde la raíz hasta el nodo hoja que se halla más a la

derecha representa la regla de clasificación: “Si una persona tiene más de 25 años, es probable que suponga un riesgo bajo para la aseguradora.”

Las reglas estructuradas en forma de árbol son muy populares, ya que resultan fáciles de interpretar. La facilidad de comprensión es muy importante, puesto que el resultado de cualquier actividad de minería de datos debe ser comprensible para los no especialistas. Además, algunos estudios han demostrado que, pese a las limitaciones de la estructura, las reglas estructuradas en forma de árbol son muy precisas. Hay algoritmos eficientes para la creación de reglas estructuradas en forma de árbol para las bases de datos de gran tamaño. En el resto de este apartado se estudia un algoritmo de ejemplo para la creación de árboles de decisión.

17.4.1 Árboles de decisión

Los árboles de decisión son representaciones gráficas de un conjunto de reglas de clasificación. Dado un registro de datos, el árbol dirige el registro desde la raíz hasta una hoja. Cada nodo interno del árbol se etiqueta con un atributo predictor. Este atributo se suele denominar **atributo divisor**, ya que los datos se “dividen” en función de las condiciones impuestas a este atributo. Los arcos salientes de cada nodo interno se etiquetan con predicados que implican al atributo divisor de ese nodo; todos los registros de datos que entran al nodo deben satisfacer el predicado y etiquetar exactamente un arco saliente. La información combinada sobre el atributo divisor y los predicados para los arcos salientes se denomina **criterio de división** del nodo. Los nodos sin arcos salientes se denominan **nodos hoja**. Cada nodo hoja del árbol se etiqueta con un valor del atributo dependiente. Sólo se tomarán en consideración árboles binarios, en los que los nodos internos tienen dos arcos salientes, aunque son posibles árboles de grados superiores.

Considérese el árbol de decisión de la Figura 17.7. El atributo divisor del nodo raíz es *edad*, el de su hijo izquierdo es *tipocoche*. El predicado del arco saliente izquierdo del nodo raíz es $edad \leq 25$, el del arco saliente derecho es $edad > 25$.

Ahora se puede asociar una regla de clasificación con cada nodo hoja del árbol de la manera siguiente. Considérese la ruta desde la raíz del árbol hasta el nodo hoja. Cada arco de esa ruta está etiquetado con un predicado. La conjunción de todos esos predicados conforma el lado izquierdo de la regla. El valor del atributo dependiente del nodo hoja constituye el lado derecho de la regla. Por tanto, el árbol de decisión representa un conjunto de reglas de clasificación, una por cada nodo hoja.

Los árboles de decisión se suelen crear en dos fases. En la primera, la **fase de crecimiento**, se crea un árbol realmente grande. Este árbol representa los registros de la base de datos de entrada con gran precisión; por ejemplo, puede que el árbol contenga nodos hoja para cada registro de la base de datos de entrada. En la fase dos, la **fase de poda**, se determina el tamaño final del árbol. Las reglas representadas por el árbol creado en la primera fase suelen estar excesivamente especializadas. Al reducir el tamaño del árbol, se crea un número menor de reglas más generales que son mejores que un número muy grande de reglas muy especializadas. Los algoritmos para la poda de árboles caen fuera del ámbito de este estudio.

Los algoritmos para los árboles de clasificación crean los árboles de forma impaciente y descendente de la manera siguiente. En el nodo raíz se examina la base de datos y se calcula

el criterio de división localmente “mejor”. A continuación, y de acuerdo con el criterio de división del nodo raíz, la base de datos se divide en dos partes: una partición para el hijo izquierdo y otra para el derecho. Este algoritmo se aplica después de manera recursiva a cada hijo. Este esquema se refleja en la Figura 17.8.

Entrada: nodo n , partición D , método de selección de división \mathcal{S}

Salida: árbol de decisión para D arrraigado en el nodo n

Esquema de inducción descendente del árbol de decisión:

Crearárbol(Nodo n , partición de datos D , método de selección de división \mathcal{S})

- (1) Aplicar \mathcal{S} a D para hallar el criterio divisor
- (2) **if** (se halla un buen criterio divisor)
- (3) Crear dos nodos hijos n_1 y n_2 de n
- (4) Dividir D en D_1 y D_2
- (5) Crearárbol(n_1 , D_1 , \mathcal{S})
- (6) Crearárbol(n_2 , D_2 , \mathcal{S})
- (7) **endif**

Figura 17.8 Esquema de inducción del árbol de decisión

El criterio divisor en cada nodo se halla mediante la aplicación de un **método de selección de división**. El método de selección de división es un algoritmo que toma como entrada (parte de) una relación y genera como salida el criterio divisor localmente “mejor”. En este ejemplo el método de selección de división examina los atributos *tipocoche* y *edad*, selecciona uno de ellos como atributo divisor y luego selecciona los predicados divisores. Se han desarrollado muchos métodos de selección de división diferentes y muy sofisticados; las referencias ofrecen indicaciones de la literatura relevante.

17.4.2 Un algoritmo para la creación de árboles de decisión

Si la base de datos de entrada cabe en memoria principal, se puede seguir directamente el esquema de inducción del árbol de clasificación mostrado en la Figura 17.8. ¿Cómo se pueden crear árboles de decisión cuando la relación de entrada es mayor que la memoria principal?

<i>edad</i>	<i>tipocoche</i>	<i>riesgoalto</i>
23	Utilitario	falso
30	Deportivo	falso
36	Utilitario	falso
25	Camión	verdadero
30	Utilitario	falso
23	Camión	verdadero
30	Camión	falso
25	Deportivo	verdadero
18	Utilitario	falso

Figura 17.9 La relación InfoSeguro

En ese caso, falla el paso (1) de la Figura 17.8, ya que la base de datos de entrada no cabe en memoria. Pero se puede hacer una observación importante sobre los métodos de selección de división que ayudará a reducir las necesidades de memoria principal.

Considérese un nodo del árbol de decisión. El método de selección de división tiene que tomar dos decisiones tras examinar la partición en ese nodo: tiene que seleccionar el atributo divisor y también los predicados divisores para los arcos salientes. Tras seleccionar el criterio de división en el nodo el algoritmo se aplica recursivamente a cada uno de los hijos de ese nodo. ¿Necesita realmente el método de selección de división toda la partición de la base de datos como entrada? Afortunadamente, la respuesta es no.

Los métodos de selección de división que calculan criterios de división que involucran a un solo atributo predictor en cada nodo evalúan cada atributo predictor por separado. Dado que cada atributo se examina por separado, se puede proporcionar el método de selección de división con información agregada sobre la base de datos en lugar de cargar toda la base de datos en memoria principal. Bien elegida, esta información agregada permite calcular el mismo criterio de división que se obtendría examinando la base de datos completa.

Dado que el método de selección de división examina todos los atributos predictores, hace falta información agregada sobre cada atributo predictor. Esta información agregada se denomina **conjunto CVA** del atributo predictor. El conjunto CVA del atributo predictor X en el nodo n es la proyección de la partición de la base de datos de n sobre X y el atributo dependiente, en la que los recuentos de los diferentes valores del dominio del atributo dependiente están agregados. (CVA significa etiqueta de Clase de los Valores del Atributo, ya que los valores del atributo dependiente se suelen denominar **etiquetas de clase**.) Por ejemplo, considérese la relación InfoSeguro tal y como aparece en la Figura 17.9. El conjunto CVA del nodo raíz del árbol para el atributo predictor $edad$ es el resultado de la siguiente consulta a la base de datos:

```
SELECT R.edad, R.riesgoalto, COUNT (*)
FROM InfoSeguro R
GROUP BY R.edad, R.riesgoalto
```

El conjunto CVA del nodo izquierdo del nodo raíz para el atributo predictor $tipocoche$ es el resultado de la consulta siguiente:

```
SELECT R.tipocoche, R.riesgoalto, COUNT (*)
FROM InfoSeguro R
WHERE R.edad <= 25
GROUP BY R.tipocoche, R.riesgoalto
```

Los dos conjuntos CVA del nodo raíz del árbol pueden verse en la Figura 17.10.

Se define el **grupo CVA** del nodo n como el conjunto de los conjuntos CVA de todos los atributos predictores en el nodo n . El ejemplo de la relación InfoSeguro tiene dos atributos predictores; por tanto, el grupo CVA de cualquier nodo consiste en dos conjuntos CVA.

¿Qué tamaño tienen los conjuntos CVA? Obsérvese que el tamaño del conjunto CVA de cada atributo predictor X en el nodo n sólo depende del número de valores diferentes del atributo de X y del tamaño del dominio del atributo dependiente. Por ejemplo, considérense los conjuntos CVA de la Figura 17.10. El conjunto CVA del atributo predictor $tipocoche$ tiene tres entradas y del atributo predictor $edad$ tiene cinco, aunque la relación InfoSeguro, tal

Tipo coche	riesgoalto	
	verdadero	falso
Utilitario	0	4
Deportivo	1	1
Camión	2	1

Edad	riesgoalto	
	verdadero	falso
18	0	1
23	1	1
25	2	0
30	0	3
36	0	1

Figura 17.10 Grupo CVA del nodo raíz de la relación InfoSeguroEntrada: nodo n , partición D , método de selección de división \mathcal{S} Salida: árbol de decisión para D con raíz n **Esquema de inducción descendente del árbol de decisión:****CrearÁrbol**(Nodo n , partición de datos D , método de selección de división \mathcal{S})(1a) Realizar un examen de D y crear en memoria el grupo CVA de n (1b) Aplicar \mathcal{S} al grupo CVA para hallar el criterio de división**Figura 17.11** Refinamiento de la inducción del árbol de clasificación con grupos CVA

y como puede verse en la Figura 17.9, tiene nueve registros. Para bases de datos de gran tamaño el tamaño de los conjuntos CVA es independiente del número de tuplas de la base de datos, excepto si hay atributos con dominios muy grandes, por ejemplo, un campo con valores reales registrado con precisión muy alta con muchas cifras tras la coma decimal.

Si se realiza la suposición simplificadora de que todos los conjuntos CVA del nodo raíz caben juntos en memoria principal, se pueden crear árboles de decisión a partir de bases de datos de tamaño muy grande de la manera siguiente: se realiza un examen de la base de datos y se crea en memoria el grupo CVA del nodo raíz. Luego se ejecuta el método de selección de división elegido con el grupo CVA como entrada. Una vez que el método de selección de división ha calculado el atributo divisor y los predicados divisores para los arcos salientes, se divide la base de datos y se vuelve a aplicar el procedimiento recursivamente. Obsérvese que este algoritmo es muy parecido al algoritmo original de la Figura 17.8; la única modificación necesaria puede verse en la Figura 17.11. Además, este algoritmo sigue siendo independiente del método de selección de división concreto utilizado.

17.5 AGRUPACIÓN

En este apartado se analiza el **problema de la agrupación**. El objetivo es dividir un conjunto de registros en grupos tales que los registros de cada grupo sean parecidos entre sí y los que pertenecen a dos grupos distintos sean diferentes. Cada uno de estos grupos se denomina **agrupación**, y cada registro pertenece exactamente a una agrupación¹. La similitud entre los registros se mide computacionalmente mediante una **función distancia**. Cada función distancia toma dos registros de entrada y devuelve un valor que es una medida

¹Hay algoritmos de agrupación que permiten agrupaciones solapadas, en las que cada registro puede pertenecer a varias agrupaciones.

de su similitud. Las diferentes aplicaciones tienen conceptos distintos de similitud y no hay ninguna medida que sea aplicable a todos los dominios.

Por ejemplo, considérese el esquema de la vista InfoCliente:

`InfoCliente(edad: int, sueldo: real)`

Se pueden dibujar los registros de la vista en un plano como puede verse en la Figura 17.12. Las dos coordenadas de cada registro son los valores de sus campos *sueldo* y *edad*. Se pueden identificar visualmente tres agrupaciones: Los clientes jóvenes que tienen sueldos bajos, los clientes jóvenes con sueldos altos y los clientes mayores con sueldos altos.

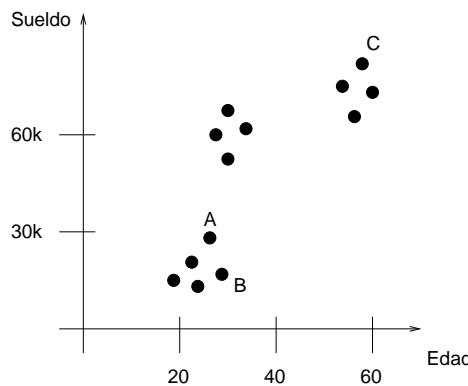


Figura 17.12 Registros de InfoCliente

Generalmente la salida de un algoritmo de agrupación consiste en una **representación resumida** de cada agrupación. El tipo de representación resumida depende mucho del tipo y de la forma de las agrupaciones que calcule el algoritmo. Por ejemplo, supóngase que se tienen agrupaciones esféricas como en el ejemplo de la Figura 17.12. Se puede resumir cada agrupación mediante su *centro* (a menudo denominado también *media*) y su *radio*. Dado el conjunto de registros r_1, \dots, r_n , su **centro** C y su **radio** R se definen de la manera siguiente:

$$C = \frac{1}{n} \sum_{i=1}^n r_i, \text{ y } R = \sqrt{\frac{\sum_{i=1}^n (r_i - C)^2}{n}}$$

Hay dos tipos de algoritmos de agrupación. Los algoritmos de agrupación **particionales** dividen los datos en k grupos tales que se optimice alguno de los criterios que evalúa la calidad de la agrupación. El número de agrupaciones k es un parámetro cuyo valor especifica el usuario. Los algoritmos de agrupación **jerárquicos** generan una secuencia de particiones de los registros. A partir de una partición en que cada agrupación consta de un solo registro, el algoritmo mezcla dos particiones en cada paso hasta que sólo quede una.

17.5.1 Un algoritmo de agrupación

La agrupación es un problema muy antiguo y se han desarrollado numerosos algoritmos para agrupar conjuntos de registros. Tradicionalmente se ha supuesto que el número de registros

de la base de datos de entrada es relativamente pequeño y que toda la base de datos cabe en la memoria principal. En este apartado se describe un algoritmo de agrupación denominado BIRCH que maneja bases de datos de tamaño muy grande. El diseño de BIRCH refleja las dos suposiciones siguientes:

- El número de registros puede ser muy grande y, por tanto, sólo se desea llevar a cabo un examen de la base de datos.
- Sólo se dispone de una cantidad limitada de memoria principal.

El usuario puede definir dos parámetros para controlar el algoritmo BIRCH. El primero es un umbral para la cantidad de memoria principal disponible. Este umbral de la memoria principal se traduce en el número máximo de resúmenes de agrupaciones k que se pueden mantener en memoria. El segundo parámetro, ϵ , es un umbral inicial para el radio de cualquier agrupación. El valor de ϵ es una cota superior para el radio de cualquier agrupación y controla el número de agrupaciones que descubre este algoritmo. Si ϵ es pequeño, se descubren muchas agrupaciones pequeñas; si ϵ es grande, se descubren muy pocas agrupaciones, cada una de las cuales es relativamente grande. Se dice que una agrupación es **compacta** si su radio es menor que ϵ .

BIRCH siempre mantiene k o menos resúmenes de agrupaciones (C_i, R_i) en la memoria principal, donde C_i es el centro de la agrupación i y R_i es su radio. El algoritmo siempre mantiene agrupaciones compactas; es decir, el radio de cada agrupación es menor que ϵ . Si esta invariante no se puede mantener con la cantidad existente de memoria principal, se incrementa ϵ como se describe a continuación.

El algoritmo lee secuencialmente registros de la base de datos y los procesa de la manera siguiente:

1. Se calcula la distancia entre el registro r y el centro de cada una de las agrupaciones existentes. Sea i el índice de las agrupaciones, de tal modo que la distancia entre r y C_i sea mínima.
2. Se calcula el valor del nuevo radio R'_i de la agrupación i -ésima suponiendo que r esté incluido en ella. Si $R'_i \leq \epsilon$, entonces la agrupación i -ésima sigue siendo compacta y se asigna r a la agrupación i -ésima actualizando su centro y definiendo su radio como R'_i . Si $R'_i > \epsilon$, la agrupación i -ésima ya no será compacta si se incluye en ella r . Por tanto, se inicia una nueva agrupación que sólo contiene el registro r .

El segundo paso presenta un problema si ya se tiene el número máximo de resúmenes de agrupaciones, k . Si ahora se lee un registro que exige que se cree una agrupación nueva, se carece de la memoria principal necesaria para albergar su resumen. En ese caso, se incrementa el umbral del radio ϵ —aplicando alguna heurística para determinar el incremento necesario—para *mezclar* las agrupaciones ya existentes. El incremento de ϵ tiene dos consecuencias. En primer lugar, las agrupaciones ya existentes pueden albergar más registros, ya que se ha incrementado su radio máximo. En segundo lugar, puede que resulte posible mezclar las agrupaciones ya existentes de tal modo que la agrupación existente siga siendo compacta. Por tanto, el incremento de ϵ suele reducir el número de agrupaciones existentes.

Todo el algoritmo BIRCH emplea un árbol equilibrado que se alberga en memoria, parecido en estructura a los árboles B+, para identificar rápidamente el centro de agrupación más

Sistemas comerciales de minería de datos. Actualmente hay en el mercado varios productos para la minería de datos, como Enterprise Miner de SAS, Clementine de SPSS, CART de Salford Systems o PolyAnalyst de Megaputer, KnowledgeStudio de ANGOSS. Se destacarán dos que tienen estrechos vínculos con las bases de datos.

Intelligent Miner de IBM ofrece una amplia gama de algoritmos, incluidas las reglas de asociación, regresión, clasificación y agrupación. El énfasis de Intelligent Miner se centra en la escalabilidad —el producto contiene versiones de todos los algoritmos para ordenadores en paralelo y está estrechamente integrado con el sistema de bases de datos DB2 de IBM—. Se pueden utilizar las posibilidades relacionales orientadas a objetos de DB2 para definir las clases de minería de datos de SQL/MM. Por supuesto, otros fabricantes de productos de minería de datos pueden utilizar esas posibilidades para añadir sus propios modelos y algoritmos de minería de datos a DB2.

SQL Server 2000 de Microsoft tiene un componente denominado Analysis Server que hace posible crear, aplicar y administrar modelos de minería de datos en el SGBD. (Las posibilidades OLAP de SQL Server también van empaquetadas en el componente Analysis Server.) El enfoque básico adoptado es representar los modelos de minería como tablas; actualmente soporta los modelos de agrupación y de árboles de decisión. Cada tabla tiene conceptualmente una fila para cada combinación posible de valores de los atributos de entrada (predictores). El modelo se crea mediante una instrucción análoga a `CREATE TABLE` de SQL que describe la entrada para la que se va a adiestrar el modelo y el algoritmo que se debe utilizar para crearlo. Una característica interesante es que se puede definir la tabla de entrada, mediante un mecanismo de vistas especializado, como *tabla anidada*. Por ejemplo, se puede definir una tabla de entrada con una fila por cliente, en la que uno de los campos sea una tabla anidada que describa las compras de ese cliente. Las extensiones SQL/MM para minería de datos no ofrecen esta posibilidad debido a que SQL:1999 no soporta en la actualidad las tablas anidadas (Apartado 15.2.1). También se pueden especificar varias propiedades de los atributos como, por ejemplo, si son discretos o continuos.

Los modelos se adiestran introduciéndoles filas mediante la orden `INSERT`. Se aplica a los conjuntos de datos nuevos para hacer predicciones mediante un nuevo tipo de reunión denominado **REUNIÓN PREDICTIVA**; en principio, cada tupla de entrada se empareja con la tupla correspondiente del modelo de minería para determinar el valor del atributo predicho. Así, los usuarios finales pueden crear, adiestrar y aplicar árboles de decisión y agrupaciones mediante SQL ampliado. También hay órdenes para explorar los modelos. Por desgracia, los usuarios no pueden añadir modelos nuevos ni algoritmos nuevos para los modelos, una posibilidad que se permite en la propuesta SQL/MM.

próximo para los registros nuevos. La descripción de esta estructura de datos cae fuera del ámbito de este análisis.

17.6 BÚSQUEDAS DE SIMILITUDES EN LAS SECUENCIAS

Gran parte de la información almacenada en bases de datos consiste en secuencias. En este apartado se presentará el problema de la búsqueda de similitudes en conjuntos de secuencias. El modelo de consulta es muy sencillo: se supone que el usuario especifica una **secuencia de consulta** y desea recuperar todas las secuencias de datos que son parecidas a la de la consulta. La búsqueda de similitudes se diferencia de las consultas “normales” en que no sólo interesan las secuencias que coinciden exactamente con la secuencia de la consulta, sino también en las que sólo se diferencian de ella ligeramente.

Se comienza describiendo las secuencias y la similitud entre secuencias. La **secuencia de datos** X es una serie de números $X = \langle x_1, \dots, x_k \rangle$. A veces X se denomina también **serie temporal**. Se dice que k es la **longitud** de la secuencia. La **subsecuencia** $Z = \langle z_1, \dots, z_j \rangle$ se obtiene de la secuencia $X = \langle x_1, \dots, x_k \rangle$ eliminando números del comienzo y del final de la secuencia X . Formalmente, Z es una subsecuencia de X si $z_1 = x_i, z_2 = x_{i+1}, \dots, z_j = x_{i+j-1}$ para algún $i \in \{1, \dots, k-j+1\}$. Dadas dos secuencias $X = \langle x_1, \dots, x_k \rangle$ e $Y = \langle y_1, \dots, y_k \rangle$, se puede definir la **norma euclídea** como la distancia entre dos secuencias de la manera siguiente:

$$\|X - Y\| = \sum_{i=1}^k (x_i - y_i)^2$$

Dada una secuencia de consulta especificada por el usuario y el parámetro de umbral ϵ , el objetivo es recuperar todas las secuencias de datos que se hallan a una distancia menor o igual que ϵ de la secuencia de consulta.

Las consultas de similitud de secuencias se pueden clasificar en dos tipos.

- **Coincidencia de secuencias completas.** La secuencia de consulta y las secuencias de la base de datos tienen la misma longitud. Dado el parámetro umbral ϵ especificado por el usuario, el objetivo es recuperar todas las secuencias de la base de datos que se hallan a una distancia menor o igual que ϵ de la secuencia de consulta.
- **Coincidencia de subsecuencias.** La secuencia de consulta es más corta que las secuencias de la base de datos. En ese caso, se desea hallar todas las subsecuencias de secuencias de la base de datos tales que esa subsecuencia se halle a distancia menor o igual que ϵ de la secuencia de consulta. No se analizará la coincidencia de subsecuencias.

17.6.1 Un algoritmo para hallar secuencias similares

Dado un conjunto de secuencias de datos, una secuencia de consulta y un umbral de distancia ϵ , ¿cómo se pueden hallar de manera eficiente todas las secuencias que se hallan a una distancia menor o igual que ϵ de la secuencia de consulta?

Una posibilidad es examinar la base de datos, recuperar todas las secuencias de datos y calcular su distancia a la secuencia de consulta. Aunque este algoritmo tiene la ventaja de ser sencillo, siempre recupera todas las secuencias de datos.

Como se considera el problema de la coincidencia de secuencias completas, todas las secuencias de datos y la secuencia de consulta tienen la misma longitud. Se puede considerar

esta búsqueda de similitudes un problema de indexación multidimensional. Las secuencias de datos y la secuencia de consulta se pueden representar como puntos en un espacio de k dimensiones. Por tanto, si se introducen todas las secuencias de datos en un índice multidimensional, se pueden recuperar secuencias de datos que coincidan exactamente con la secuencia de consulta mediante la consulta del índice. Pero, dado que no sólo se desea recuperar secuencias de datos que coincidan exactamente con la de consulta, sino también todas las secuencias a una distancia menor o igual que ϵ de la secuencia de consulta, no se emplean consultas puntuales tal y como las define la secuencia de consulta. En cambio, se consulta el índice con un hiperrectángulo que tiene un lado de longitud 2ϵ y la secuencia de consulta como centro y se recuperan todas las consultas que se hallan dentro de ese hiperrectángulo. Luego se descartan las secuencias que se hallen realmente a una distancia mayor que ϵ de la secuencia de consulta.

El empleo del índice permite reducir enormemente el número de secuencias que se consideran y disminuye de manera significativa el tiempo necesario para evaluar la consulta de similitud. Las notas bibliográficas del final de este capítulo ofrecen indicaciones de mejoras ulteriores.

17.7 MINERÍA INCREMENTAL Y CORRIENTES DE DATOS

Los datos de la vida real no son estáticos, sino que evolucionan constantemente mediante la inserción o eliminación de registros. En algunas aplicaciones, como la vigilancia de la red, los datos llegan en corrientes de velocidad tan alta que no resulta posible almacenarlos para su análisis fuera de línea. Se describirán tanto los datos en evolución como las corrientes de datos en términos de una infraestructura denominada **evolución de bloques**. En la evolución de bloques el conjunto de datos de entrada para el proceso de minería de datos no es estático, sino que se actualiza de manera periódica con nuevos bloques de tuplas, por ejemplo, todos los días a medianoche o mediante una corriente continua. Cada **bloque** es un conjunto de tuplas añadidas de manera simultánea a la base de datos. Para bloques de gran tamaño, este modelo captura la práctica habitual en muchas de las instalaciones actuales de almacenes de datos, en las que las actualizaciones procedentes de las bases de datos operativas se agrupan y se llevan a cabo en bloque. Para los bloques de datos de pequeño tamaño —en el extremo, cada bloque consiste en un solo registro— este modelo captura corrientes de datos.

En el modelo de evolución de bloques la base de datos consiste en una (teóricamente infinita) secuencia de bloques de datos B_1, B_2, \dots que llega en los momentos $1, 2, \dots$, en los que cada bloque B_i consiste en un conjunto de registros². Se denomina i al *identificador de bloque* del bloque B_i . Por tanto, en cualquier momento m , la base de datos consiste en una secuencia finita de bloques de datos $\langle B_1, \dots, B_m \rangle$ que han llegado en los momentos $\{1, 2, \dots, m\}$. La base de datos en el momento m , que se denota mediante $D[1, m]$, es la unión de las base de datos en el momento $m - 1$ y el bloque que ha llegado en el momento m (B_m).

²En general, cada bloque especifica los registros que hay que modificar o eliminar, además de los registros que hay que insertar. Aquí sólo se considerarán las inserciones.

Para los datos en evolución dos clases de problemas resultan de especial interés: el mantenimiento de modelos y la detección de cambios. El objetivo del **mantenimiento de modelos** es mantener el modelo de minería de datos pese a las inserciones y eliminaciones de bloques de datos. Para calcular de manera incremental el modelo de minería de datos en el momento m , que se denota mediante $M(D[1, m])$, sólo hay que tomar en consideración $M(D[1, m - 1])$ y B_m ; no se pueden tomar en consideración los datos que llegaron antes del momento m . Además, los analistas de datos pueden especificar subconjuntos de $D[1, m]$ dependientes del tiempo, como los intervalos de interés (por ejemplo, todos los datos vistos hasta ahora o los datos de la semana pasada). También son posibles selecciones más generales, por ejemplo, todos los datos de los fines de semana del año pasado. Dadas esas selecciones, hay que calcular el modelo de manera incremental para el subconjunto de datos correspondiente de $D[1, m]$ considerando sólo D_m y el modelo para el subconjunto correspondiente de $D[1, m - 1]$. Los algoritmos “casi” incrementales que examinan a veces datos más antiguos podrían resultar aceptables en aplicaciones de almacenes de datos, en las que la incrementalidad viene motivada por consideraciones de eficiencia y los datos antiguos están disponibles si hacen falta. Esta opción no está disponible para las corrientes de datos de alta velocidad, en las que puede que los datos antiguos no estén disponibles en absoluto.

El objetivo de la **detección de cambios** es cuantificar la diferencia, en términos de las características de los datos, entre dos conjuntos de datos y determinar si el cambio es significativo (es decir, estadísticamente significativo). En concreto, hay que cuantificar la diferencia entre los modelos de datos tal y como existían en el momento m_1 y la versión evolucionada en un momento posterior m_2 ; es decir, hay que cuantificar la diferencia entre $M(D[1, m_1])$ y $M(D[1, m_2])$. También se pueden medir los cambios con respecto a subconjuntos seleccionados de datos. Hay varias variantes naturales del problema; por ejemplo, la diferencia entre $M(D[1, m - 1])$ y $M(D_m)$ indica si el último bloque se diferencia de manera sustancial de los datos existentes previamente. En el resto de este capítulo nos centraremos en el mantenimiento de modelos y no se estudiará la detección de cambios.

El mantenimiento incremental de los modelos ha recibido mucha atención. Dado que la calidad de los modelos de minería de datos es de suprema importancia, los algoritmos de mantenimiento incremental de los modelos se han concentrado en el cálculo exacto del mismo modelo que se obtiene ejecutando el algoritmo básico de creación del modelo sobre la unión de los datos viejos con los nuevos. Una técnica de escalabilidad muy utilizada es la localización de las modificaciones debidas a los bloques nuevos. Por ejemplo, para los algoritmos de agrupación basados en la densidad, la inclusión de registros nuevos sólo afecta a las agrupaciones que se hallan en la proximidad de esos registros y, por tanto, los algoritmos eficientes pueden *localizar* el cambio en unas pocas agrupaciones y evitar volver a calcularlas todas. Como ejemplo adicional, en la creación de árboles de decisión puede que se tenga la posibilidad de mostrar que el criterio de división en un nodo del árbol sólo cambia dentro de intervalos de confianza aceptablemente pequeños cuando se insertan registros, si se supone que la distribución subyacente de registros de adiestramiento es estática.

La creación de modelos de paso único sobre corrientes de datos ha recibido especial atención, ya que los datos llegan y deben procesarse continuamente en varios dominios de aplicación emergentes. Por ejemplo, las instalaciones de red de los grandes proveedores de servicios de telecomunicaciones y de Internet tienen información de uso detallada (por ejemplo, registros de detalles de llamadas, flujo de paquetes por los enruteadores y datos de seguimiento)

de diferentes partes de sus redes subyacentes que se deben analizar de manera continua para detectar tendencias interesantes. Otros ejemplos son los registros de los servidores Web, las corrientes de datos de transacciones de grandes cadenas de venta minorista y las cotizaciones financieras.

Al trabajar con corrientes de datos de alta velocidad se deben diseñar los algoritmos para que creen los modelos de minería de datos mientras examinan los elementos de datos relevantes *sólo una vez y en un orden preestablecido* (determinado por la pauta de llegada de la corriente), con una cantidad limitada de memoria principal. El cálculo con corrientes de datos ha dado lugar a varios estudios recientes (tanto teóricos como prácticos) de los algoritmos en línea o de paso único con memoria limitada. Se han desarrollado algoritmos para el cálculo en un solo paso de cuantiles y de estadísticas de orden, la estimación de momentos de frecuencia y de tamaños de reunión, la creación de agrupaciones y de árboles de decisión, la estimación de los agregados correlacionados y el cálculo de histogramas monodimensionales (es decir, con un solo atributo) y de descomposiciones en ondas de Haar. A continuación se examinará uno de estos algoritmos para el mantenimiento incremental de lotes frecuentes.

17.7.1 Mantenimiento incremental de lotes frecuentes

Considérese la relación Compras de la Figura 17.1 y supóngase que el umbral de soporte mínimo es el 60%. Se puede ver fácilmente que el conjunto de lotes frecuentes de tamaño 1 consiste en $\{pluma\}$, $\{tinta\}$ y $\{leche\}$, con soportes del 100%, el 75% y el 75%, respectivamente. El conjunto de lotes frecuentes de tamaño 2 consiste en $\{pluma, tinta\}$ y $\{pluma, leche\}$, los dos con soportes del 75%. La relación Compras es el primer bloque de datos. El objetivo es desarrollar un algoritmo que conserve el conjunto de lotes frecuentes ante la inserción de nuevos bloques de datos.

Como primer ejemplo, considérese la inserción del bloque de datos de la Figura 17.13 a la base de datos original (Figura 17.1). Con esta inserción, el conjunto de lotes frecuentes no cambia, aunque sí lo hace el valor de sus soportes: $\{pluma\}$, $\{tinta\}$ y $\{leche\}$ tienen ahora valores de soporte del 100%, el 60% y el 60%, respectivamente, y $\{pluma, tinta\}$ y $\{pluma, leche\}$ tienen ahora un soporte del 60%. Obsérvese que se podría detectar este caso de “no cambio” con sólo mantener el número de cestas de la compra en las que aparece cada lote. En este ejemplo se actualiza el soporte (absoluto) del lote $\{pluma\}$ en 1.

<i>idtrans</i>	<i>idcli</i>	<i>fecha</i>	<i>artículo</i>	<i>cantidad</i>
115	201	7/1/99	pluma	2

Figura 17.13 El bloque 2 de la relación Compras

En general, el conjunto de lotes frecuentes puede cambiar. Por ejemplo, considérese la inserción del bloque de la Figura 17.14 a la base de datos original mostrada en la Figura 17.1. Se puede ver una transacción que contiene el artículo *agua*, pero no se conoce el soporte del lote $\{agua\}$, ya que el agua no superaba el soporte mínimo exigido en la base de datos original. Una solución sencilla de este caso es realizar un examen adicional de la base de datos original y calcular luego el soporte del lote $\{agua\}$. Pero ¿se puede hacer aún mejor?

<i>idtrans</i>	<i>idcli</i>	<i>fecha</i>	<i>artículo</i>	<i>cantidad</i>
115	201	7/1/99	agua	1
115	201	7/1/99	leche	1

Figura 17.14 El bloque 2a de la relación Compras

Otra solución inmediata es tener contadores para *todos* los lotes posibles, pero el número de lotes posibles es exponencial en el número de artículo —y, en todo caso, la mayoría de estos contadores sería 0—. ¿Se puede diseñar una estrategia inteligente que indique *qué* contadores considerar?

Se introducirá el concepto de **frontera negativa** de los conjuntos de lotes para ayudar a decidir qué contadores se deben considerar. La frontera negativa de un conjunto de lotes frecuentes consiste en todos los lotes *X* tales que el propio *X* no sea frecuente, pero que sí lo sean todos los subconjuntos de *X*. Por ejemplo, en el caso de la base de datos de la Figura 17.1, los lotes siguientes constituyen la frontera negativa: $\{zumo\}$, $\{agua\}$ y $\{tinta, leche\}$. Ahora se puede diseñar un algoritmo más eficiente para el mantenimiento de lotes frecuentes si se tienen contadores para todos los lotes frecuentes actuales *y* para todos los lotes que se hallan en este momento en la frontera negativa. Sólo si algún lote de la frontera negativa pasa a ser frecuente hay que volver a leer la base de datos original, para averiguar el soporte de los lotes nuevos que pudieran pasar a ser frecuentes.

Esta afirmación se ilustra mediante los dos ejemplos siguientes. Si se añade el Bloque 2a de la Figura 17.14 a la base de datos original de la Figura 17.1, se aumenta el soporte del lote frecuente $\{leche\}$ en uno y el del lote $\{agua\}$, que se halla en la frontera negativa, también en uno. Pero, dado que ningún lote de la frontera negativa ha pasado a ser frecuente, no hace falta volver a examinar la base de datos original.

Por el contrario, considérese la inserción del Bloque 2b de la Figura 17.15 a la base de datos original de la Figura 17.1. En ese caso, el lote $\{zumo\}$, que se hallaba en un principio en la frontera negativa, pasa a ser frecuente, con un soporte del 60%. Esto significa que ahora los siguientes lotes de tamaño dos entran en la frontera negativa: $\{zumo, pluma\}$, $\{zumo, tinta\}$ y $\{zumo, leche\}$. ($\{zumo, agua\}$ no puede ser frecuente, ya que el lote $\{agua\}$ no es frecuente).

<i>idtrans</i>	<i>idcli</i>	<i>fecha</i>	<i>artículo</i>	<i>cantidad</i>
115	201	7/1/99	zumo	2
115	201	7/1/99	agua	2

Figura 17.15 El Bloque 2b de la relación Compras

17.8 OTRAS TAREAS EN LA MINERÍA DE DATOS

Este capítulo se ha centrado en el problema del descubrimiento de pautas en bases de datos, pero hay otras tareas de minería de datos igual de importantes. Ahora se indicarán brevemente

algunas de ellas. Las notas bibliográficas del final del capítulo ofrecen muchas referencias para estudios ulteriores.

- **Selección de conjuntos de datos y de sus características.** A menudo es importante seleccionar el conjunto de datos “adecuado” para la minería. La selección de conjuntos de datos es el proceso de hallar los conjuntos de datos en los que llevar a cabo la minería de datos. La selección de características es el proceso de decidir los atributos que se deben incluir en el proceso de minería.
- **Muestreo.** Un modo de explorar conjuntos de datos de gran tamaño es obtener una o varias *muestras* y analizarlas. La ventaja del muestreo es que se pueden llevar a cabo análisis detallados de muestras que resultarían imposibles para todo el conjunto de datos, en el caso de conjuntos de datos muy grandes. El inconveniente del muestreo es que resulta difícil la obtención de muestras representativas para cada tarea; puede que se pasen por alto tendencias o pautas importantes debido a que no se hallan reflejadas en la muestra. Los sistemas de bases de datos actuales ofrecen también poco soporte a la obtención eficiente de muestras. La mejora del soporte a las bases de datos para la obtención de muestras con diferentes propiedades estadísticas deseables resulta sencilla y es posible que se halle disponible en SGBD futuros. La aplicación del muestreo a la minería de datos es un área de investigación adicional.
- **Visualización.** Las técnicas de visualización pueden ayudar de manera significativa a comprender los conjuntos de datos complejos y a la detección de pautas interesantes, y la importancia de la visualización en la minería de datos está ampliamene reconocida.

17.9 PREGUNTAS DE REPASO

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

- ¿Cuál es el papel de la minería de datos en el proceso KDD? (**Apartado 17.1**)
- ¿Qué es la propiedad *a priori*? Descríbase un algoritmo para la búsqueda de lotes frecuentes. (**Apartado 17.2.1**)
- ¿Qué relación tienen las consultas iceberg con los lotes frecuentes? (**Apartado 17.2.2**)
- Dé la definición de *regla de asociación*. ¿Cuál es la diferencia entre el soporte y la confianza de una regla? (**Apartado 17.3.1**)
- ¿Explíquense las ampliaciones de las reglas de asociación para las jerarquías ES? ¿Con qué otras ampliaciones de las reglas de asociación está familiarizado? (**Apartado 17.3.3 y 17.3.4**)
- ¿Qué es una pauta secuencial? ¿Cómo se pueden calcular las pautas secuenciales? (**Apartado 17.3.5**)
- ¿Se pueden utilizar las reglas de asociación para realizar predicciones? (**Apartado 17.3.6**)

- ¿Cuál es la diferencia entre las redes bayesianas y las reglas de asociación? (**Apartado 17.3.7**)
- ¿Puede dar ejemplos de reglas de clasificación y de regresión? ¿Cómo se definen el soporte y la confianza para esas reglas? (**Apartado 17.3.8**)
- ¿Cuáles son los componentes de los árboles de decisión? ¿Cómo se crean los árboles de decisión? (**Apartados 17.4.1 y 17.4.2**)
- ¿Qué es una agrupación? ¿Qué información se suele obtener de las asociaciones? (**Apartado 17.5**)
- ¿Cómo se puede definir la distancia entre dos secuencias? Describase un algoritmo que busque todas las secuencias parecidas a una secuencia de consultas dada. (**Apartado 17.6**)
- Describase el modelo de evolución de bloques y defínanse los problemas del mantenimiento incremental del modelo y la detección de cambios. ¿Cuál es el desafío añadido de la minería de corrientes de datos? (**Apartado 17.7**)
- Describase un algoritmo incremental para el cálculo de lotes frecuentes. (**Apartado 17.7.1**)
- Dense ejemplos de otras tareas relacionadas con la minería de datos. (**Apartado 17.8**)

EJERCICIOS

Ejercicio 17.1 Respóndase brevemente a las siguientes preguntas:

1. Defínase *soporte* y *confianza* para las reglas de asociación.
2. Explíquese el motivo de que las reglas de asociación no se puedan utilizar directamente para la predicción, sin análisis ulterior o conocimiento del dominio.
3. ¿Cuáles son las diferencias entre las *reglas de asociación*, las *reglas de clasificación* y las *reglas de regresión*?
4. ¿Cuál es la diferencia entre *clasificación* y *agrupación*?
5. ¿Cuál es el papel de la visualización de la información en la minería de datos?
6. Dense ejemplos de consultas a una base de datos de cotizaciones de acciones, almacenadas como secuencias, una por acción, que no se puedan expresar en SQL.

Ejercicio 17.2 Considérese la tabla Compras de la Figura 17.1.

1. Simúlese el algoritmo para la búsqueda de lotes frecuentes en la tabla de la Figura 17.1 con *sopmín*=90% y luego hállense las reglas de asociación con *confmín*=90%.
2. ¿Se puede modificar la tabla de modo que los mismos lotes frecuentes se obtengan con *sopmín*=90% y con *sopmín*=70% en la tabla de la Figura 17.1?
3. Simúlese el algoritmo para la búsqueda de lotes frecuentes en la tabla de la Figura 17.1 con *sopmín*=10% y luego hállense las reglas de asociación con *confmín*=90%.
4. ¿Se puede modificar la tabla de modo que los mismos lotes frecuentes se obtengan con *sopmín*=10% y con *sopmín*=70% en la tabla de la Figura 17.1?

<i>idtrans</i>	<i>idcli</i>	<i>fecha</i>	<i>artículo</i>	<i>cant</i>
111	201	5/1/2002	tinta	1
111	201	5/1/2002	leche	2
111	201	5/1/2002	zumo	1
112	105	6/3/2002	pluma	1
112	105	6/3/2002	tinta	1
112	105	6/3/2002	agua	1
113	106	5/10/2002	pluma	1
113	106	5/10/2002	agua	2
113	106	5/10/2002	leche	1
114	201	6/1/2002	pluma	2
114	201	6/1/2002	tinta	2
114	201	6/1/2002	zumo	4
114	201	6/1/2002	agua	1
114	201	6/1/2002	leche	1

Figura 17.16 La relación Compras2

Ejercicio 17.3 Supóngase que se tiene el conjunto de datos C de cestas de la compra y se ha calculado el conjunto de lotes frecuentes \mathcal{X} de C para un umbral de soporte dado $sopmín$. Supóngase que se quisiera añadir otro conjunto de datos C' a C y conservar el conjunto de lotes frecuentes con el umbral de soporte $sopmín$ en $C \cup C'$. Considérese el algoritmo siguiente para el mantenimiento incremental del conjunto de lotes frecuentes:

1. Se ejecuta el algoritmo *a priori* en C' y se hallan todos los lotes frecuentes de C' y su soporte correspondiente. El resultado es el conjunto de lotes \mathcal{X}' . También se calcula el soporte de todos los lotes $X \in \mathcal{X}$ en C' .
2. Luego se realiza un examen de C para calcular el soporte de todos los lotes en \mathcal{X}' .

Respóndanse las preguntas siguientes sobre este algoritmo:

- Falta el último paso del algoritmo; es decir, ¿cuál será el resultado del algoritmo?
- ¿Es más eficiente este algoritmo que los algoritmos descritos en el Apartado 17.7.1?

Ejercicio 17.4 Considérese la tabla Compras2 de la Figura 17.16.

- Indíquense todos los lotes de la frontera negativa del conjunto de datos.
- Indíquense todos los lotes frecuentes con un umbral de soporte del 50%.
- Dese un ejemplo de base de datos en el que la inserción en esa base de datos no modifique esta frontera negativa.
- Dese un ejemplo de base de datos en el que la inserción en esa base de datos modifique esta frontera negativa.

Ejercicio 17.5 Considérese la tabla Compras de la Figura 17.1. Hállense todas las reglas de asociación (generalizadas) que indiquen la probabilidad de que un mismo cliente compre ciertos artículos en la misma fecha, con $sopmín$ del 10% y $confmín$ del 70%.

Ejercicio 17.6 Desarrollemos un nuevo algoritmo para el cálculo de todos los lotes de gran tamaño. Supóngase que tenemos la relación R parecida a la tabla Compras de la Figura 17.1. Dividimos horizontalmente la tabla en k partes R_1, \dots, R_k .

1. Pruébese que, si el lote X es frecuente en R , también es frecuente, como mínimo, en una de sus k partes.
2. Utilícese esta información para desarrollar un algoritmo que calcule todos los lotes frecuentes con dos exámenes de R . (*Sugerencia:* en el primer examen se deben calcular los lotes localmente frecuentes para cada parte R_i , $i \in \{1, \dots, k\}$.)
3. Ilústrese el algoritmo mediante la tabla Compras de la Figura 17.1. La primera división consiste en las dos transacciones con *idtrans* 111 y 112, la segunda consiste en las dos transacciones con *idtrans* 113 y 114. Supóngase que el soporte mínimo es del 70%.

Ejercicio 17.7 Considérese la tabla Compras de la Figura 17.1. Hállese todas las pautas secuenciales con *sopmín* del 60%. (El texto sólo esboza el algoritmo para la búsqueda de pautas secuenciales, por lo que se debe emplear la fuerza bruta o leer alguna de las referencias bibliográficas para conseguir un algoritmo completo.)

edad	sueldo	suscripción
37	45k	No
39	70k	Sí
56	50k	Sí
52	43k	Sí
35	90k	Sí
32	54k	No
40	58k	No
55	85k	Sí
43	68k	Sí

Figura 17.17 La relación InfoSuscriptor

Ejercicio 17.8 Considérese la relación InfoSuscriptor de la Figura 17.17. Contiene información sobre la campaña de marketing de la revista *Radioaficionado*. Las dos primeras columnas muestran la edad y el sueldo de los posibles clientes y la columna *suscripción* muestra si esas personas están suscritas a la revista. Se desea utilizar estos datos para crear un árbol de decisiones que ayude a predecir si una persona dada se suscribirá a la revista.

1. Créese el grupo CVA del nodo raíz del árbol.
2. Supóngase que el predicado divisor del nodo raíz es $edad \leq 50$. Créense los grupos CVA de los dos nodos hijo del nodo raíz.

Ejercicio 17.9 Supóngase que se tiene el siguiente conjunto de seis registros: $\langle 7, 55 \rangle$, $\langle 21, 202 \rangle$, $\langle 25, 220 \rangle$, $\langle 12, 73 \rangle$, $\langle 8, 61 \rangle$ y $\langle 22, 249 \rangle$.

1. Suponiendo que los seis registros pertenecen a una sola agrupación, calcúlense su centro y su radio.
2. Supóngase que los tres primeros registros pertenecen a una agrupación y los tres últimos a otra diferente. Calcúlense el centro y el radio de las dos agrupaciones.
3. ¿Cuál de las dos agrupaciones es “mejor” y por qué?

Ejercicio 17.10 Supóngase que se tienen las tres secuencias $\langle 1, 3, 4 \rangle$, $\langle 2, 3, 2 \rangle$ y $\langle 3, 3, 7 \rangle$. Calcúlese la norma euclídea de todos los pares de secuencias.

NOTAS BIBLIOGRÁFICAS

Entre los textos introductorios a la minería de datos figuran [237, 243, 317, 476].

El descubrimiento de conocimiento útil a partir de bases de datos de gran tamaño supera la mera aplicación de un conjunto de algoritmos de minería de datos, y el punto de vista de que se trata de un proceso iterativo guiado por un analista se subraya en [172] y [400]. El trabajo en el análisis explorador de datos en estadística, por ejemplo, [453] y en aprendizaje de la máquina y en descubrimiento de conocimiento en inteligencia artificial fueron precursores de la atención actual a la minería de datos; el énfasis añadido en los grandes volúmenes de datos es el elemento nuevo de importancia. Entre los buenos resúmenes recientes de la minería de datos figuran [260, 174, 313]. [173] contiene resúmenes adicionales y artículos sobre muchos aspectos de la minería de datos y del descubrimiento de conocimiento, incluido un tutorial sobre redes bayesianas [244]. El libro de Piatetsky-Shapiro y Frawley [358] y el de Fayyad, Piatetsky-Shapiro, Smyth y Uthurusamy [175] contienen colecciones de trabajos sobre minería de datos. La conferencia anual SIGKDD, organizada por el grupo de trabajo de la ACM para el descubrimiento de conocimiento en bases de datos, es un buen recurso para los lectores [176, 416, 245, 16, 101, 368] interesados en la investigación más reciente en minería de datos [176, 416, 245, 16, 101, 368], al igual que el *Journal of Knowledge Discovery and Data Mining*.

El problema de la obtención de reglas de asociación lo introdujeron Agrawal, Imielinski y Swami [11]. Se han propuesto muchos algoritmos eficientes para el cálculo de lotes de gran tamaño, como [12, 69, 238, 410, 447].

Las consultas iceberg las introdujeron Fang *et al.* [171]. También hay gran cantidad de investigación sobre las formas generalizadas de las reglas de asociación; por ejemplo, [423, 424, 426]. El problema de la búsqueda de los lotes frecuentes máximos también ha recibido una atención significativa [226, 227, 45, 75, 478, 299, 8]. Los algoritmos para la obtención de reglas de asociación con restricciones se consideran en [426, 343, 290, 46, 355, 356].

Los algoritmos paralelos se describen en [14] y en [391]. Se pueden hallar trabajos recientes sobre la minería paralela de datos en [479], y el trabajo sobre la minería distribuida de datos se puede hallar en [266].

[191] presenta un algoritmo para la búsqueda de reglas de asociación en atributos numéricos continuos; las reglas sobre los atributos numéricos se estudian también en [477]. La forma general de las reglas de asociación, en la que los atributos distintos del identificador de transacción se agrupan, se desarrolla en [329]. Las reglas de asociación para los artículos de una jerarquía se estudian en [423, 235]. Se proponen más ampliaciones y generalizaciones de las reglas de asociación en [67, 343, 45]. La integración de la minería para los lotes frecuentes en los sistemas de bases de datos se aborda en [390, 451]. El problema de la obtención de pautas secuenciales se analiza en [15], y se pueden encontrar más algoritmos para la obtención de pautas secuenciales en [316, 425].

Se pueden encontrar introducciones generales a las reglas de clasificación y de regresión en [236, 332]. La referencia clásica para la creación de árboles de decisiones y de regresión es el libro CART de Friedman, Olshen y Stone [66]. Quinlan ofrece una perspectiva del aprendizaje de la máquina sobre la creación de árboles de decisiones [362]. Recientemente se han desarrollado varios algoritmos para la creación de árboles de decisiones [202, 204, 324, 372, 403].

El problema de las agrupaciones se ha estudiado durante décadas en varias disciplinas. Entre los libros de texto figuran [154, 261, 267]. Entre los algoritmos de agrupación escalables tenemos CLARANS [342], DBSCAN [159, 160], BIRCH [488] y CURE [224]. Bradley, Fayyad y Reina abordan el problema del escalado del algoritmo de agrupación de las medias K para bases de datos de gran tamaño [65, 64]. El problema de la búsqueda de agrupaciones en subconjuntos de los campos se aborda en [10]. Ganti *et al.* examinan el problema de la agrupación de datos en espacios métricos arbitrarios [198]. Entre los algoritmos para la agrupación de datos categóricos figuran STIRR [208] y CACTUS [197]. [387] es un algoritmo de agrupación para datos espaciales.

La búsqueda de secuencias parecidas en grandes bases de datos de secuencias se estudia en [13, 170, 284, 364, 408].

El trabajo en el mantenimiento incremental de reglas de asociación se considera en [109, 110, 445]. Ester *et al.* describen la manera de mantener agrupaciones de modo incremental [158], mientras que Hidber describe el modo de hacer lo mismo con lotes de gran tamaño [250]. También ha habido trabajo reciente en la minería de corrientes de datos, como puede ser la construcción de árboles de decisión [152, 257, 202] y de agrupaciones

586 Sistemas de gestión de bases de datos

de corrientes de datos [223, 344]. En [195] se presenta un marco general para la minería de datos en evolución. En [196] se propone un marco para la medida de las modificaciones de las características de los datos.



18

RECUPERACIÓN DE INFORMACIÓN Y DATOS XML

- ¿Cómo evolucionan los SGBD en respuesta a la creciente cantidad de datos de texto?
- ¿Qué es el modelo de espacio vectorial y cómo soporta la búsqueda de texto?
- ¿Cómo se indexan los conjuntos de texto?
- Comparado con los sistemas de RI, ¿qué hay de novedoso en la búsqueda Web?
- ¿En qué se diferencian los datos XML del texto sencillo y de las tablas relacionales?
- ¿Cuáles son las características principales de XQuery?
- ¿Cuáles son los desafíos de implementación que plantean los datos XML?
- ➡ **Conceptos principales:** recuperación de información, consultas booleanas y clasificadas; relevancia, precisión, recuperación; modelo de espacio vectorial, peso de términos FT/FID, similitud entre documentos; índice inverso, archivo de firmas; rastreador Web, nodos y autoridades, clasificación de las páginas Web; modelo de datos semiestructurados, XML; XQuery, expresiones de ruta, consultas FLWR; almacenamiento e indexación XML.

en colaboración con Raghav Kaushik
Universidad de Wisconsin-Madison

Un *memex* es un dispositivo en el que las personas guardan todos sus libros, discos y comunicaciones y que se mecaniza de modo que se pueda consultar con velocidad y flexibilidad extremas.

— Vannevar Bush, *Cómo podríamos pensar, 1945*

El campo de la **recuperación de información (RI)** ha estudiado el problema de la búsqueda en conjuntos de documentos de texto desde los años cincuenta del siglo veinte y se ha

desarrollado en su mayor parte independientemente de los sistemas de bases de datos. La proliferación de documentos de texto en Web ha hecho de la búsqueda en documentos una operación cotidiana para la mayor parte de la gente y ha motivado un renovado interés por la información en este asunto.

El deseo de quienes trabajan en el campo de las bases de datos de ampliar los tipos de datos que se pueden gestionar en SGBD está bien documentado y se refleja en desarrollos como las extensiones relacionales orientadas a objetos (Capítulo 15). Los documentos en Web representan una de las fuentes de datos de crecimiento más rápido y el desafío de su gestión en SGBD se ha transformado de manera natural en un punto focal de la investigación en bases de datos.

Por tanto, Web ha acercado más que nunca los campos de los sistemas de administración de bases de datos y de la recuperación de información y, como se verá, XML se halla justo en un terreno intermedio entre ambos campos. A continuación se introducirán los sistemas de RI, así como un modelo de datos y un lenguaje de consulta para los datos XML, y se analizará la relación con los sistemas de bases de datos orientados a objetos.

En este capítulo se presenta una visión general de la recuperación de información, de la búsqueda Web y del modelo de datos XML y de las normas para los lenguajes de consulta. Se comenzará en el Apartado 18.1 con un estudio del modo en que encajan estas tendencias orientadas a texto en el contexto de los sistemas de bases de datos orientados a objetos actuales. Se presentarán conceptos de recuperación de información en el Apartado 18.2 y se examinarán técnicas especializadas de indexación para texto en el Apartado 18.3. Los motores de búsqueda Web se estudian en el Apartado 18.4. En el Apartado 18.5 se describen brevemente las tendencias actuales de ampliación de los sistemas de bases de datos para que soporten datos de texto y se identificarán algunos de los principales problemas implicados. En el Apartado 18.6 se presenta el modelo de datos de XML, conociendo los conceptos de XML presentados en el Capítulo 7. Se describe el lenguaje XQuery en el Apartado 18.7. En el Apartado 18.8 se considera la evaluación eficiente de consultas en XQuery.

18.1 MUNDOS EN CONFLICTO: BASES DE DATOS, RI Y XML

La red Web es hoy en día el conjunto de documentos más ampliamente utilizado, y la búsqueda Web se diferencia en aspectos importantes de la recuperación de documentos tradicional al estilo RI. En primer lugar, hay un gran énfasis en la escalabilidad a conjuntos muy grandes de documentos. Los sistemas de RI suelen tratar con decenas de miles de documentos, mientras que Web contiene miles de millones de páginas.

En segundo lugar, Web ha cambiado significativamente el modo en que se crean y utilizan los conjuntos de documentos. Tradicionalmente, los sistemas de RI se dirigían a profesionales como los bibliotecarios y los investigadores legales, que estaban formados en el empleo de motores de recuperación sofisticados. Los documentos se preparaban cuidadosamente, y los de cada conjunto solían ser de temas relacionados. En Web los documentos los crea una infinita variedad de individuos debido a otras tantas razones y reflejan esa diversidad en su tamaño y su contenido. Las búsquedas las llevan a cabo personas normales sin formación en el empleo de software de recuperación.

La aparición de XML ha añadido una tercera e interesante dimensión a la búsqueda de texto: ahora todos los documentos se pueden marcar para que reflejen información adicional de interés, como la autoría, el origen e, incluso, detalles sobre su contenido intrínseco. Esto ha transformado la naturaleza de los “documentos” de texto libre a objetos textuales con campos asociados que contienen **metadatos** (datos sobre datos) o información descriptiva. Los vínculos con otros documentos son un tipo especialmente importante de metadatos y pueden tener gran valor para la búsqueda de conjuntos de documentos en Web.

La red Web también ha cambiado el concepto de lo que constituye un documento. Los documentos Web pueden ser objetos multimedia como las imágenes o los videoclips, en los que el texto sólo aparece en las etiquetas descriptivas. Se deben poder administrar esos datos tan heterogéneos y permitir búsquedas en ellos.

Los sistemas de administración de bases de datos han tratado tradicionalmente con datos tabulares sencillos. En los últimos años se han diseñado sistemas de bases de datos relacionales orientados a objetos (SGBDROO) para soportar tipos de datos complejos. Las imágenes, los videos y los objetos de texto se han mencionado explícitamente como ejemplos de los tipos de datos que se pretende que soporten los SGBDROO. No obstante, los sistemas de bases de datos actuales tienen ante sí un gran recorrido antes de que puedan soportar satisfactoriamente tipos de datos tan complejos. En el contexto de los datos de texto y de XML figuran entre los desafíos el soporte eficiente de las búsquedas en el contenido textual y de las búsquedas que aprovechan la estructura laxa de los datos XML.

18.1.1 SGBD y sistemas de RI

Los sistemas de bases de datos y de RI tienen el objetivo compartido de soportar búsquedas en conjuntos de datos. Sin embargo, muchas diferencias importantes han influido en su desarrollo.

- **Búsquedas y consultas.** Los sistemas de RI están diseñados para soportar un tipo especializado de consultas que también se denominan **búsquedas**. Las búsquedas se especifican en términos de unos pocos **términos de búsqueda**, y los datos subyacentes suelen ser un conjunto de documentos de texto sin estructurar. Además, una característica importante de las búsquedas de RI es que sus resultados se pueden **clasificar**, u ordenar, en términos de lo “bien” que coinciden con los términos de la búsqueda. Por el contrario, los sistemas de bases de datos soportan un tipo de consultas muy general, y los datos subyacentes están rígidamente estructurados. A diferencia de los sistemas de RI, los sistemas de bases de datos han venido devolviendo tradicionalmente conjuntos de resultados sin clasificar. (Ni siquiera las recientes extensiones SQL/OLAP que soportan resultados precoces y búsquedas en datos ordenados [véase el Capítulo 16] ordenan los resultados en términos de su coincidencia con la consulta formulada. Las consultas relacionales son *precisas* en el sentido de que cada fila se halla en la respuesta o no; no existe el concepto de “lo bien que coincide” con la consulta.) En otros términos, las consultas relacionales sólo asignan dos clasificaciones a las filas, que indican si la fila se halla en la respuesta o no.
- **Actualizaciones y transacciones.** Los sistemas de RI están optimizados para cargas de trabajo que consisten sobre todo en operaciones de lectura y no soportan el concepto de transacción. En los sistemas de RI tradicionales se añaden documentos nuevos al

conjunto de documentos cada cierto tiempo, y las estructuras de índices que aceleran las búsquedas se reconstruyen o actualizan de manera periódica. Por tanto, puede que haya en el sistema de RI documentos muy relevantes para una búsqueda dada que no se puedan recuperar todavía debido a unas estructuras de índices desactualizadas. Por el contrario, los sistemas de bases de datos están diseñados para manejar una amplia gama de cargas de trabajo, incluidas las cargas de trabajo de transacciones intensivas en actualizaciones.

Estas diferencias en los objetivos de diseño han llevado, lógicamente, a énfasis en la investigación y a diseños de sistemas muy diferentes. La investigación en RI ha estudiado ampliamente las funciones de clasificación. Por ejemplo, entre otros asuntos, la investigación en RI ha tratado la manera de incorporar las respuestas del comportamiento de los usuarios para modificar las funciones de clasificación, y el modo de aplicar técnicas de procesamiento lingüístico para mejorar las búsquedas. La investigación en bases de datos se ha concentrado en el procesamiento de las consultas, el control de la concurrencia y la recuperación, así como en otros temas, como se trata en este libro.

Las diferencias entre los SGBD y los sistemas de RI desde el punto de vista del diseño y de la implementación deberían quedar claras tras la introducción de los sistemas de RI en los próximos apartados.

18.2 INTRODUCCIÓN A LA RECUPERACIÓN DE INFORMACIÓN

Hay dos tipos frecuentes de búsquedas, o consultas, en los conjuntos de texto: las consultas booleanas y las clasificadas. En las **consultas booleanas**, el usuario especifica una expresión construida mediante términos y los operadores booleanos (**And**, **Or**, **Not**). Por ejemplo,

base de datos And (Microsoft Or IBM)

Esta consulta pide todos los documentos que contienen el término *base de datos* y, además, bien *Microsoft* o bien *IBM*.

En las **consultas de clasificación** el usuario especifica uno o más términos y el resultado de la consulta es una lista de documentos clasificados por su relevancia para la consulta. De manera intuitiva, se espera que los documentos de la parte superior de la lista de resultados “coincidan” más con la condición de búsqueda o sean “más relevantes” que los que se hallan más abajo en la lista de resultados. Aunque los documentos que contienen *Microsoft* satisfacen la búsqueda “*Microsoft, IBM*”, se considera que los que también contienen *IBM* son una mejor coincidencia. De manera parecida, puede que los documentos que contienen varias apariciones de *Microsoft* sean una mejor coincidencia que los que contienen una sola aparición. La clasificación de los documentos que satisfacen la condición booleana de búsqueda es un aspecto importante de los motores de búsqueda de RI, y la manera en que esto se lleva a cabo se analiza en los Apartados 18.2.3 y 18.4.2.

Una extensión importante de las consultas de clasificación es la petición de documentos que resulten más relevantes para una frase de lenguaje natural dada. Dado que las frases tienen una estructura lingüística (por ejemplo, las relaciones entre sujeto, verbo y objetos),

iddoc	Documento
1	agente James Bond buen agente
2	agente móvil ordenador
3	James Madison película
4	James Bond película

Figura 18.1 Base de datos de texto con cuatro registros

iddoc	agente	Bond	ordenador	buen	James	Madison	móvil	película
1	2	1	0	1	1	0	0	0
2	1	0	1	0	0	0	1	0
3	0	0	0	0	1	1	0	1
4	0	1	0	0	1	0	0	1

Figura 18.2 Vectores documentales para el conjunto de ejemplo

ofrecen más información que la mera lista de las palabras que contienen. No se analizará la **búsqueda en lenguaje natural**.

18.2.1 El modelo del espacio vectorial

A continuación se describirá un entramado muy empleado para la representación de documentos y la búsqueda en conjuntos de documentos. Considérese el conjunto de todos los términos que aparecen en un conjunto de documentos dados. Cada documento se puede representar como un vector con una entrada por término. En la forma más sencilla de los vectores documentales, si el término j aparece k veces en el documento i , el **vector documental** para el documento i contiene el valor k en la posición j . El vector documental para i contiene el valor 0 en las posiciones correspondientes a los términos que no aparecen en i .

Considérese el conjunto de ejemplo de cuatro documentos que puede verse en la Figura 18.1. La representación de vector documental se ilustra en la Figura 18.2; cada fila representa un documento. Esta representación de los documentos como vectores de términos se denomina **modelo del espacio vectorial**.

18.2.2 Peso de los términos según la FT/FID

Se ha descrito el valor de cada término en los vectores documentales sencillamente como la **frecuencia del término (FT)**, o número de apariciones de ese término en el documento dado. Esto refleja la intuición de que los términos que aparecen a menudo son más importantes para la caracterización de documentos que los que sólo aparecen una vez (o los que no aparecen en absoluto).

No obstante, algunos términos aparecen muy frecuentemente en el conjunto de documentos y otros son relativamente raros. Se ha observado que la frecuencia de los términos sigue una

distribución Zipf, como se ilustra en la Figura 18.3. En esta figura cada posición del eje X corresponde a un término y la del eje Y corresponde al número de apariciones de ese término. Los términos se distribuyen en el eje X en orden decreciente del número de veces que aparecen (en el global del conjunto de documentos).

Como cabría esperar, resulta que los términos extremadamente frecuentes no resultan muy útiles en las búsquedas. Ejemplos de esos términos frecuentes son *un*, *una*, *el*, etc. Los términos que aparecen muy a menudo se denominan **palabras de parada**, y los documentos se preprocesan para eliminarlos.

Incluso tras la eliminación de las palabras de parada se produce el fenómeno de que algunas palabras aparecen en el conjunto de documentos mucho más a menudo que otras. Considerense las palabras *Linux* y *núcleo* en el contexto de un conjunto de documentos sobre el sistema operativo Linux. Aunque ninguna es lo bastante frecuente como para ser palabra de parada, es probable que *Linux* aparezca mucho más a menudo. Dada una búsqueda que contenga estas dos palabras clave, es probable que se obtengan mejores resultados si se da más importancia a los documentos que contengan *núcleo* que a los que contengan *Linux*.

Se puede capturar esta intuición refinando la representación mediante vectores documentales de la manera siguiente. El valor asociado con el término j en el vector documental para el documento i , denotado como p_{ij} , se obtiene mediante la multiplicación de la frecuencia del término t_{ij} (el número de veces que el término j aparece en el documento i) por la **frecuencia inversa en el documento (FID)** del término j en el conjunto de documentos. La FID del término j se define como $\log(N/n_j)$, donde N es el número total de documentos y n_j es el número de documentos en los que aparece el término j . Esto incrementa de manera efectiva el peso dado a los términos poco frecuentes. Por ejemplo, en un conjunto de 10.000 documentos, un término que aparezca en la mitad de los documentos tendrá una FID de 0,3 y otro que sólo aparezca en un documento tendrá una FID de 4.

Normalización de longitudes

Considerese un documento D . Supóngase que se modifica añadiéndole gran número de términos nuevos. ¿Debe ser igual el peso del término t que aparece en D en el vector documental de D que en el del documento modificado? Aunque el peso FT/FID de t sigue siendo el mismo en los dos vectores documentales, la intuición sugiere que el peso debería ser menor en el documento modificado. Los documentos más largos suelen tener más términos, y más apariciones de cada término. Por tanto, si dos documentos contienen el mismo número de apariciones de un término dado, la importancia del término para la caracterización de cada uno de ellos dependerá también de la longitud de cada documento.

Se han propuesto varios enfoques de la **normalización de longitudes**. De manera intuitiva, todos ellos reducen la importancia dada a la frecuencia de aparición de los términos a medida que esa frecuencia aumenta. En los sistemas tradicionales de RI, una manera popular de refinar la métrica de similitudes es la **normalización de longitudes mediante cosenos**:

$$p_{ij}^* = \frac{p_{ij}}{\sqrt{\sum_{k=1}^t p_{ik}^2}}$$

En esta fórmula t es el número de términos del conjunto de documentos, p_{ij} es el peso FT/FID sin normalización de longitudes y p_{ij}^* es el peso FT/FID ajustado por la longitud.

Los términos que aparecen con frecuencia en los documentos resultan especialmente problemáticos en Web, ya que a menudo se modifican las páginas Web de manera deliberada añadiéndoles muchas copias de determinadas palabras —por ejemplo, rebajas, gratis, sexo— para aumentar las posibilidades de que se devuelvan como respuesta a consultas. Por este motivo, los motores de búsqueda de Web suelen normalizar por la longitud mediante la imposición de un valor máximo (normalmente 2 o 3) para la frecuencia de los términos.

18.2.3 Clasificación de documentos por su similitud

Ahora se considerará la manera en que la representación del espacio vectorial permite clasificar los documentos en los resultados de las consultas de clasificación. Una observación fundamental es que se puede considerar a las propias consultas de clasificación como documentos, ya que no son más que conjuntos de términos. Esto permite emplear la **similitud entre documentos** como base para clasificar los resultados de las consultas —al documento que es más parecido a la consulta se le da una mejor posición y al que es menos parecido se le da una peor—.

Si aparece un total de t términos en el conjunto de documentos (t es 8 en el ejemplo de la Figura 18.2), se pueden visualizar vectores documentales en un espacio t -dimensional en el que cada eje está etiquetado con un término. Esto se ilustra en la Figura 18.4 para un espacio bidimensional. La figura muestra los vectores documentales de dos documentos, D_1 y D_2 , así como para la consulta C .

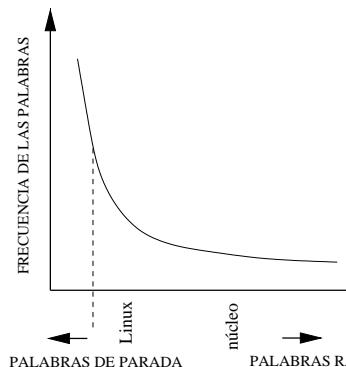


Figura 18.3 Distribución Zipf de la frecuencia de los términos

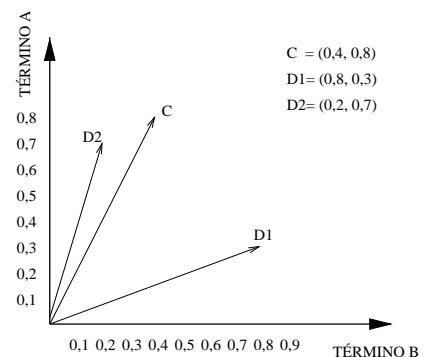


Figura 18.4 Similitud entre documentos

Se emplea la medida tradicional de la cercanía de dos vectores, su *producto escalar*, como medida de la similitud entre documentos. La similitud de la consulta C con el documento D_i se mide mediante su producto escalar:

$$\text{sim}(C, D_i) = \sum_{j=1}^t q_j^* \cdot p_{ij}^*$$

En el ejemplo de la Figura 18.4, $sim(C, D_1) = (0,4 * 0,8) + (0,8 * 0,3) = 0,56$, y $sim(C, D_2) = (0,4 * 0,2) + (0,8 * 0,7) = 0,64$. En consecuencia, D_2 se clasifica por encima de D_1 en el resultado de la búsqueda.

En el contexto Web la similitud entre documentos es una de las medidas que se pueden utilizar para clasificar los resultados, pero no se debe emplear de manera exclusiva. En primer lugar, resulta dudoso si los usuarios desean documentos que sean parecidos a la consulta (que suele constar de una o dos palabras) o documentos que contengan información útil relativa a los términos de la consulta. De manera intuitiva, se desea dar importancia a la *calidad* de las páginas Web cuando al clasificarlas, además de reflejar su parecido con una consulta dada. Los vínculos entre páginas ofrecen valiosa información adicional que se puede utilizar para conseguir resultados de gran calidad. Este asunto se examina en el Apartado 18.4.2.

18.2.4 Medida del éxito: precisión y recuperación

Se suelen emplear dos criterios para evaluar los sistemas de recuperación de información. La **precisión** es el porcentaje de documentos recuperados que son relevantes para la consulta. La **recuperación** es el porcentaje de documentos relevantes de la base de datos que se recuperan en respuesta a una consulta.

La recuperación de todos los documentos en respuesta a una consulta garantiza de manera trivial una recuperación perfecta, pero ofrece una precisión muy baja. El reto es conseguir una buena recuperación junto con una precisión elevada.

En el contexto de la búsqueda Web, el tamaño del conjunto subyacente es del orden de miles de millones de documentos. Por tanto, resulta cuestionable si la medida tradicional de la recuperación resulta muy útil. Dado que los usuarios no suelen examinar más allá de la primera pantalla de resultados, la calidad de los motores de búsqueda Web viene muy determinada por los resultados que aparecen en la primera página. Puede que las siguientes definiciones adaptadas de precisión y de recuperación resulten más adecuadas para los motores de búsqueda Web:

- **Precisión en la búsqueda Web.** El porcentaje de resultados de la primera página que resultan relevantes para la consulta.
- **Recuperación en la búsqueda Web.** La proporción N/M , expresada como porcentaje, donde M es el número de resultados mostrados en la primera página y, de los M documentos más relevantes, N es el número que aparece en la primera página.

18.3 INDEXACIÓN PARA LA BÚSQUEDA DE TEXTO

En este apartado se presentan dos técnicas de indexación que soportan la evaluación de consultas booleanas y de consultas de clasificación. La estructura de *índice invertido* examinada en el Apartado 18.3.1 se utiliza mucho debido a su sencillez y a su buen rendimiento. Su principal inconveniente es que impone una sobrecarga de espacio significativa: el tamaño puede alcanzar el triple del archivo original. El índice de *archivo de firmas* examinado en el Apartado 18.3.2 tiene una pequeña sobrecarga de espacio y ofrece un filtro rápido que elimina la mayor parte de los documentos que no cumplen las condiciones especificadas. Sin embargo,

no se adapta igual de bien a las bases de datos de mayor tamaño, ya que este índice hay que explorarlo secuencialmente.

Antes de indexar un documento, se suele preprocesar para eliminar las palabras de parada. Dado que el tamaño de los índices es muy sensible al número de términos del conjunto de documentos, la eliminación de las palabras de parada puede reducirlo enormemente. Los sistemas de RI también llevan a cabo otros tipos de preprocesamiento. Por ejemplo, aplican las **raíces** para reducir los términos relacionados a una forma canónica. Este paso reduce también el número de términos que hay que indexar pero, lo que es igual de importante, permite recuperar documentos que puede que no contengan la consulta exacta sino alguna variante de la misma. Por ejemplo, los términos *correr*, *corriendo* y *corredor* proceden todos de *correr*. Se indexa el término *correr*, y todas las apariciones de variantes de este término se tratan como apariciones de *correr*. Las consultas que especifiquen *corredor* hallarán documentos que contienen cualquier palabra que proceda de *correr*.

18.3.1 Índices invertidos

Un **índice invertido** es una estructura de datos que permite la recuperación rápida de todos los documentos que contienen términos de la consulta. Para cada término el índice mantiene una lista (denominada **lista invertida**) de entradas que describen las apariciones de ese término, con una entrada por cada documento que lo contenga.

Considérese el índice invertido del ejemplo que estamos utilizando, que puede verse en la Figura 18.5. El término “James” tiene una lista invertida con una entrada para cada uno de los documentos 1, 3 y 4; el término “agente” tiene entradas para los documentos 1 y 2.

La entrada del documento *d* en la lista invertida para el término *t* contiene detalles sobre las apariciones del término *t* en el documento *d*. En la Figura 18.5 esta información consiste en una lista de lugares dentro del documento que contienen el término *t*. Así, la entrada para el documento 1 en la lista invertida para el término “agente” muestra los lugares 1 y 5, ya que “agente” es la primera y la quinta palabra del documento 1. En general, se puede guardar en la lista invertida información adicional sobre cada aparición (por ejemplo, en documentos HTML, ¿la aparición se produce en la etiqueta TITLE?). También se puede guardar la longitud del documento si se emplea para normalizar por longitud (ver más adelante).

El conjunto de listas invertidas se denomina **archivo de publicaciones**. Las listas invertidas pueden ser muy grandes para los conjuntos de documentos de gran tamaño. De hecho, los motores de búsqueda Web suelen guardar cada lista invertida en una página distinta, y la mayor parte de las listas abarcan varias páginas (en ese caso, se mantienen en forma de lista de páginas enlazadas). Con objeto de hallar rápidamente la lista invertida de un término de consulta dado todos los términos de consulta posibles se organizan en una segunda estructura de índices, como pueden ser los árboles B+ o los índices asociativos.

Ese segundo índice, denominado **diccionario**, es mucho más pequeño que el archivo de publicaciones, ya que sólo contiene una entrada por término y, además, sólo contiene entradas para el conjunto de términos que se conserva tras eliminar las palabras de parada y aplicar las reglas de las raíces. Cada entrada consta de un término, de una información breve sobre su lista invertida y de la dirección (en disco) de la lista invertida. En la Figura 18.5 la información breve consiste en el número de entradas de la lista invertida (es decir, el número

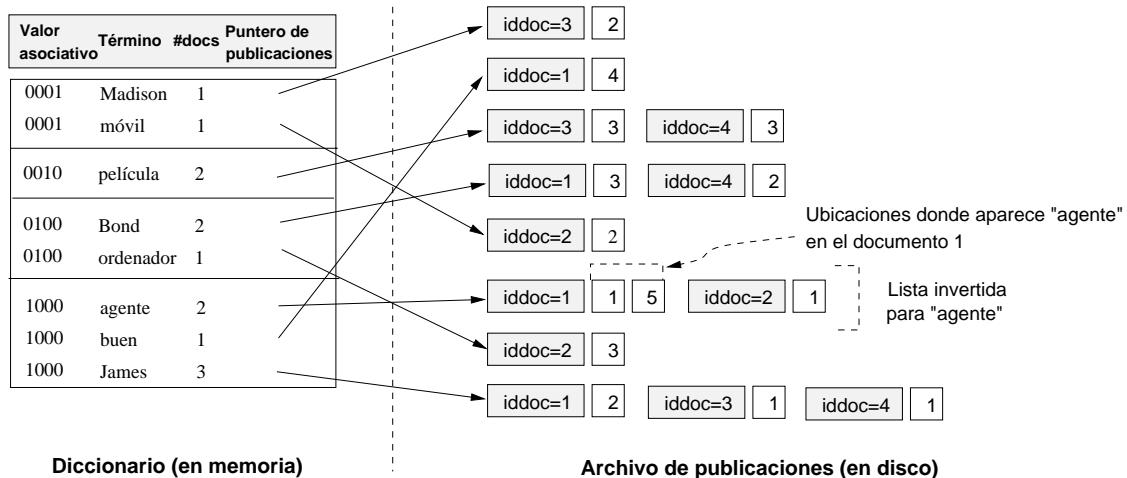


Figura 18.5 Índice invertido para el conjunto de ejemplo

de documentos en los que aparece ese término). En general, puede contener información adicional, como la FID del término, pero es importante que el tamaño de cada entrada se mantenga tan pequeño como sea posible.

El diccionario se conserva en memoria y permite la recuperación rápida de la lista invertida de los términos de las consultas. El diccionario de la Figura 18.5 emplea un índice asociativo, y se esboza mostrando el valor asociativo del término; las entradas de los términos se agrupan en cubos asociativos de acuerdo con su valor de asociación.

Empleo de índices invertidos

Las consultas que contienen un solo término se evalúan buscando primero en el diccionario la dirección de la lista invertida del término correspondiente. Luego se recupera la lista invertida, se relacionan las iddocs que figuran en ella con direcciones físicas de documentos y se recuperan los documentos correspondientes. Si hay que clasificar los resultados, se calcula la relevancia de cada documento de la lista invertida para el término de consulta y luego se recuperan los documentos por orden de la relevancia de su clasificación. Obsérvese que la información necesaria para calcular las medidas de la relevancia descritas en el Apartado 18.2 —la frecuencia del término de consulta en el documento, la FID del término en el conjunto de documentos y la longitud del documento si se emplea para la normalización por longitudes— está disponible en el diccionario o en la lista invertida.

Cuando las listas invertidas son muy largas, como ocurre con los motores de búsqueda Web, resulta útil considerar si se debe calcular previamente la relevancia de los documentos de la lista invertida de cada término (con respecto a ese término) y ordenar la lista de acuerdo con la relevancia en vez de por el identificador de documento. Esto aceleraría la consulta, ya que bastaría con examinar el prefijo de la lista invertida, puesto que los usuarios rara vez leen más allá de los primeros resultados. Sin embargo, el mantenimiento de listas ordenadas por relevancia puede resultar costoso. (La ordenación por identificador de documento resulta

cómoda, ya que se asignan identificadores crecientes a los documentos nuevos y, por tanto, basta con añadir entradas para ellos al final de la lista invertida. Además, si se modifica la función de similitud, no hace falta reconstruir el índice.)

Las consultas con la conjunción de varios términos se evalúan recuperando las listas invertidas de los términos de consulta una a una y calculando su intersección. Con objeto de minimizar el empleo de memoria, las listas invertidas se deben recuperar por orden de longitud creciente. Las consultas con disyunciones de varios términos se evalúan uniendo todas las listas invertidas relevantes.

Considérese el índice invertido de ejemplo mostrado en la Figura 18.5. Para evaluar la consulta “James” se busca en el diccionario la dirección de la lista invertida de “James”, se captura en el disco y se recupera el documento 1. Para evaluar la consulta “James” AND “Bond” hay que recuperar primero la lista invertida del término “Bond” y calcular su intersección con la del término “James”. (La lista invertida del término “Bond” tiene longitud dos, mientras que la de “James” tiene longitud tres.) El resultado de la intersección de la lista $\langle 1, 4 \rangle$ con la lista $\langle 1, 3, 4 \rangle$ es la lista $\langle 1, 4 \rangle$ y, por tanto, se recuperan los documentos 1 y 4. Para evaluar la consulta “James” OR “Bond” se recuperan las dos listas invertidas en el orden que se desee y se unen los resultados.

Para las consultas de clasificación con varios términos hay que capturar las listas invertidas de todos los términos, calcular la relevancia de cada documento que aparezca en alguna de ellas con respecto al conjunto de términos de búsqueda dado y ordenar los identificadores de los documentos de acuerdo con su clasificación de relevancia antes de capturar los documentos en ese mismo orden. Nuevamente, si se ordenan las listas invertidas según alguna medida de relevancia, se pueden soportar consultas de clasificación procesando normalmente sólo prefijos de pequeño tamaño de las listas invertidas. (Obsérvese que la relevancia de los documentos con respecto a las consultas se calcula fácilmente a partir de su relevancia con respecto a cada término de consulta.)

18.3.2 Archivos de firmas

Los **archivos de firmas** son otra estructura de índice para los sistemas de bases de datos de texto que soporta la evaluación eficiente de consultas booleanas. Cada archivo de firmas contiene un registro de índice para cada documento de la base de datos. Ese registro de índice se denomina **firma** del documento. Cada firma tiene un tamaño fijo de b bits; b se denomina **anchura de firmas**. Los bits asignados dependen de las palabras que aparecen en el documento. Las palabras se relacionan con los bits mediante la aplicación de una función de asociación a cada palabra del documento y se asignan los bits que aparecen en el resultado de esa función. Obsérvese que, a menos que se tenga un bit por cada palabra posible del vocabulario, se puede asignar dos veces el mismo bit a palabras diferentes, ya que la función de asociación asignará ambas palabras al mismo bit. Se dice que la firma F_1 coincide con la firma F_2 si todos los bits asignados en F_2 se asignan también en F_1 . Si la firma F_1 coincide con la firma F_2 , entonces F_1 tiene, como mínimo, tantos bits asignados como F_2 .

Para consultas que consisten en una conjunción de términos se genera en primer lugar la firma de la consulta aplicando la función asociativa a cada palabra de la consulta. Luego se explora el archivo de firmas y se recuperan todos los documentos cuyas firmas coincidan

iddoc	Documento	Firma
1	agente James Bond buen agente	1100
2	agente ordenador portátil	1101
3	James Madison película	1011
4	James Bond película	1110

Figura 18.6 Archivo de firmas del conjunto de ejemplo

con la firma de la consulta, ya que todos esos documentos son resultados potenciales de la consulta. Dado que la firma no identifica las palabras que contiene cada documento de manera unívoca, hay que recuperar todas las posibles coincidencias y comprobar si el documento contiene realmente los términos de la consulta. Los documentos cuya firma coincide con la de la consulta pero que no contienen todos sus términos se denominan **falsos positivos**. Los falsos positivos son errores costosos, ya que hay que recuperar el documento del disco, analizarlo, buscar las raíces y comprobarlo para determinar si contiene los términos de la consulta.

Para consultas que consisten en disyunciones de términos se genera una lista de firmas de la consulta, una para cada término de la consulta. La consulta se evalúa explorando el archivo de firmas para hallar los documentos cuyas firmas coinciden con alguna de las de la lista de firmas de la consulta.

Por ejemplo, considérese el archivo de firmas de anchura 4 del ejemplo de la Figura 18.6. Los bits asignados por los valores asociados a todos los términos de la consulta se muestran en la figura. Para evaluar la consulta “James” se calcula primero el valor asociado al término; es 1000. Luego se explora el archivo de firmas y se buscan los registros de índice coincidentes. Como puede verse en la Figura 18.6, las firmas de todos los registros tienen asignado el primer bit. Se recuperan todos los documentos y se buscan falsos positivos; el único falso positivo para esta consulta es el documento con identificador 2. (Por desgracia, da la casualidad de que el valor asociado al término “agente” también se asignó al primer bit de la firma.) Considérese la consulta “James” And “Bond”. La firma de la consulta es 1100 y tres firmas de documento coinciden con ella. Una vez más, se recupera un falso positivo. Como ejemplo adicional de consulta conjuntiva, considérese la consulta “película” And “Madison”. La firma de la consulta es 0011, y sólo una firma de documento coincide con ella. No se recupera ningún falso positivo.

Obsérvese que para cada consulta hay que explorar todo el archivo de firmas, y que hay tantos registros en él como documentos en la base de datos. Para reducir la cantidad de datos que hay que recuperar para cada consulta se puede dividir verticalmente para cada consulta el archivo de firmas en un conjunto de **cortes de bits** y denominar a ese índice **archivo de firmas de cortes de bits**. La longitud de cada corte de bits sigue siendo igual al número de documentos de la base de datos, pero para una consulta con q bits asignados en la firma sólo hace falta recuperar q cortes de bits. Se invita al lector a crear un archivo de firmas de cortes de bits y a evaluar las consultas de ejemplo de este párrafo mediante las cortes de bits.

18.4 MOTORES DE BÚSQUEDA WEB

Los motores de búsqueda Web deben enfrentarse a un número de documentos extraordinariamente grande y tienen que ser muy escalables. Los documentos también están vinculados entre sí, y resulta que la información de esos vínculos es muy valiosa para hallar las páginas relevantes para una búsqueda dada. Estos factores han hecho que los motores de búsqueda se diferencien de los sistemas de RI tradicionales en aspectos importantes. No obstante, confían en alguna modalidad de índices invertidos como mecanismo básico de indexación. En este apartado se examinarán los motores de búsqueda Web, empleando a Google como ejemplo típico.

18.4.1 Arquitectura de los motores de búsqueda

Los motores de búsqueda Web **rastrean** la red Web para recoger documentos a indexar. El algoritmo de rastreo es sencillo, pero el software rastreador puede ser complejo, debido a los detalles de las conexiones con millones de sitios, la minimización de la latencia de las redes, la ejecución en paralelo del rastreo, la resolución de la superación de los tiempos máximos de espera y de otros fallos de las conexiones, la necesidad de garantizar que los sitios que se examinan no sean sometidos a una tensión indebida por el rastreador y a otros problemas prácticos.

El algoritmo de búsqueda empleado por el rastreador es un recorrido de grafos. Partiendo de un conjunto de páginas con muchos vínculos (por ejemplo, las páginas de directorio de Yahoo), se siguen todos los vínculos de las páginas analizadas para identificar otras nuevas. Este paso se itera, mientras se conserva un registro de las páginas que se han visitado con objeto de evitar volver a visitarlas.

El conjunto de páginas recuperadas mediante el rastreo puede ser enorme, del orden de miles de millones de páginas. Su indexación es una tarea costosa. Por fortuna, gran parte de esa tarea se puede realizar en paralelo: cada documento se analiza de manera independiente para crear listas invertidas para los términos que aparecen en él. Esas listas realizadas para cada documento se ordenan luego por términos y se unen para crear listas invertidas completas para cada término que abarcan todos los documentos. Las estadísticas de los términos, como la FID, se pueden calcular durante la fase de fusión.

El soporte de búsquedas en índices tan grandes es otra empresa gigantesca. Por fortuna, una vez más, la tarea se puede realizar fácilmente en paralelo mediante agrupaciones de máquinas de bajo coste: se puede trabajar con esa cantidad de datos dividiendo el índice entre varias máquinas. Cada máquina contiene el índice invertido de aquellos términos que le han sido asignados (por ejemplo, asociando el término). Puede que haya que enviar cada consulta a varias máquinas si los términos que contiene los tratan máquinas diferentes pero, dado que las consultas de Web rara vez contienen más de dos términos, en la práctica no se trata de un problema grave.

También hay que trabajar con un volumen enorme de consultas; Google soporta más de ciento cincuenta millones de búsquedas cada día, y esa cifra sigue creciendo. Esto se consigue replicando los datos en varias máquinas. A cada partición se le asignan ahora varias máquinas, cada una de las cuales contiene una copia exacta de los datos de esa partición. Las consultas

a esa partición puede tratarlas cualquier máquina de la misma. Las consultas se pueden distribuir entre varias máquinas de acuerdo con la carga de trabajo, asociando por dirección IP, etcétera. La réplica también aborda el problema de la elevada disponibilidad, ya que el fallo de una máquina sólo aumenta la carga de las demás de esa partición y, si cada partición contiene varias máquinas, el impacto es pequeño. Los fallos pueden hacerse transparentes a los usuarios mediante el encaminamiento de las consultas a otras máquinas mediante el equilibrador de cargas.

18.4.2 Empleo de la información sobre los vínculos

Gran variedad de usuarios crea páginas Web por motivos muy diversos, y su contenido no siempre se presta a una recuperación efectiva. Puede que las páginas más relevantes para una búsqueda no contengan ninguno de los términos de búsqueda y, por tanto, no los devuelva ninguna búsqueda booleana de palabras clave. Por ejemplo, considérese el término de búsqueda “navegador Web”. Una consulta de texto booleana que emplee esos términos no devolverá las páginas relevantes de Netscape Corporation ni de Microsoft, ya que esas páginas no contienen el término “navegador Web”. De manera parecida, la página inicial de Yahoo no contiene el término “motor de búsqueda”. El problema es que los sitios relevantes no necesariamente describen su contenido de manera que sea útil para las consultas de texto booleanas.

Hasta ahora se ha considerado únicamente la información que se halla dentro de una sola página Web para estimar su relevancia para una consulta dada. Pero las páginas Web están conectadas mediante hipervínculos, y es bastante posible que haya una página que contenga el término “motor de búsqueda” que tenga un vínculo a la página inicial de Yahoo. ¿Se puede utilizar la información oculta en esos vínculos?

Aprovechando la investigación en la literatura sociológica, una analogía interesante entre los vínculos y las citas bibliográficas sugiere una manera de aprovechar la información de los vínculos: igual que los autores y las publicaciones influyentes se citan a menudo, es probable que las buenas páginas Web aparezcan a menudo en los vínculos. Resulta útil distinguir dos tipos de páginas, *autoridades* y *nodos*. Las **autoridades** son páginas que resultan muy importantes para un asunto determinado y que están reconocidas por otras páginas como expertas en ese tema. Esas otras páginas, denominadas nodos, suelen tener un número significativo de hipervínculos con las autoridades, aunque ellas mismas no sean muy bien conocidas y no tengan necesariamente mucho contenido relevante para la consulta dada. Las páginas **nodo** pueden ser compilaciones de recursos sobre un tema en un sitio para profesionales, listas de sitios recomendados para las aficiones de un usuario concreto o, incluso, parte de las páginas favoritas de un usuario dado que son relevantes para alguno de sus intereses; su principal propiedad es que tienen muchos vínculos salientes con páginas relevantes. A menudo, las buenas páginas nodo no son muy conocidas y puede que haya pocos vínculos que apunten a un buen nodo. Por el contrario, las buenas autoridades son “avaladas” por muchos buenos nodos y, por tanto, tienen muchos vínculos desde buenas páginas nodo.

Esta relación simbiótica entre nodos y autoridades es la base del algoritmo HITS, un algoritmo de búsqueda basado en los vínculos que descubre páginas de gran calidad que son relevantes para los términos de búsqueda de los usuarios. El algoritmo HITS modela

Cálculo de los pesos de nodo y autoridad. Se puede emplear la notación matricial para escribir las actualizaciones de todos los pesos de nodo y de autoridad en un paso dado. Supóngase que se numeran todas las páginas del conjunto base como $\{1, 2, \dots, n\}$. La matriz de adyacencia B del conjunto base es una matriz $n \times n$ cuyos elementos son 0 o 1. El elemento (i, j) de la matriz se define como 1 si la página i tiene un hipervínculo con la página j ; en caso contrario, se define como 0. También se pueden escribir los pesos de nodo n y los pesos de autoridad a en notación vectorial: $n = \langle h_1, \dots, h_n \rangle$ y $a = \langle a_1, \dots, a_n \rangle$. Ahora se pueden volver a escribir las reglas de actualización de la manera siguiente:

$$n = B \cdot a, \quad y \quad a = B^T \cdot n.$$

Si se desarrolla esta ecuación una vez, lo que corresponde a la primera notación, se obtiene:

$$n = BB^T n = (BB^T)n, \quad y \quad a = B^T Ba = (B^T B)a.$$

Tras la segunda iteración se llega a:

$$n = (BB^T)^2 n, \quad y \quad a = (B^T B)^2 a.$$

Los resultados del álgebra lineal indican que la secuencia de iteraciones para los pesos de nodo (o de autoridad) converge a los autovectores principales de BB^T (o $B^T B$) si se normalizan los pesos antes de cada iteración de modo que la suma de los cuadrados de todos los pesos sea siempre $2 \cdot n$. Además, los resultados del álgebra lineal indican que esa convergencia es independiente de la elección de los pesos iniciales, siempre que sean positivos. Por tanto, nuestra elección un tanto arbitraria de pesos iniciales —se asignó a todos los pesos de nodo y de autoridad un valor inicial de 1— no modifica el resultado del algoritmo.

Web como si fuera un grafo dirigido. Cada página Web representa un nodo del grafo, y el hipervínculo de la página A a la página B se representa como un arco entre los dos nodos correspondientes.

Supóngase una consulta de usuario con varios términos. El algoritmo procede en dos pasos. En el primero, el *paso de muestreo*, se reúne un conjunto de páginas denominado **conjunto base**. Lo más probable es que el conjunto base incluya páginas muy relevantes para la consulta del usuario, pero puede seguir siendo bastante grande. En el segundo paso, el *paso de iteración*, se hallan buenas autoridades y buenos nodos entre las páginas del conjunto base.

El paso de muestreo recupera un conjunto de páginas Web que contienen los términos de la consulta empleando alguna técnica tradicional. Por ejemplo, se puede evaluar la consulta como la búsqueda booleana de una palabra clave y recuperar todas las páginas Web que contienen los términos de la consulta. Al conjunto de páginas resultante se le denomina **conjunto raíz**. Puede que el conjunto raíz no contenga todas las páginas relevantes porque alguna página con autoridad no incluya las palabras de la consulta del usuario. Pero se espera que, como mínimo, alguna de las páginas del conjunto raíz contenga hipervínculos con las páginas con autoridad más relevantes o que algunas páginas con autoridad enlacen con

La clasificación paloma de Google (Pigeon Rank). Google calcula la *clasificación paloma (CP)* de la página Web A mediante la fórmula siguiente, que es muy parecida a las funciones de clasificación Nodo-Autoridad:

$$CP(A) = (1 - d) + d(CP(T_1)/C(T_1) + \dots + CP(T_n)/C(T_n))$$

$T_1 \dots T_n$ son las páginas que enlazan (o “apuntan a”) con A , $C(T_i)$ es el número de vínculos salientes de la página T_i y d es una constante elegida de manera heurística (Google emplea 0,85). Las clasificaciones paloma forman una distribución de probabilidad para todas las páginas Web; la suma de las clasificaciones para todas las páginas es 1. Si se considera un modelo de comportamiento de los usuarios en el que un usuario escoge una página al azar y luego pulsa repetidamente los vínculos hasta que se aburre y escoge otra página al azar, la probabilidad de que ese usuario visite una página es su clasificación paloma. Las páginas del resultado de una búsqueda se clasifican mediante una combinación de métricas de relevancia al estilo RI y de clasificaciones paloma.

páginas del conjunto raíz. Esto justifica el concepto de **página de enlace**. Una página se denomina de enlace si tiene algún hipervínculo a páginas del conjunto raíz o si alguna página del conjunto raíz tiene hipervínculos con ella. Para no pasar por alto páginas potencialmente relevantes, se incrementa el conjunto raíz con todas las páginas de enlace y se denomina al conjunto de páginas resultante **conjunto base**. Por tanto, el conjunto base incluye todas las páginas raíz y todas las páginas de enlace; nos referiremos a las páginas Web del conjunto base como **páginas base**.

El objetivo del segundo paso del algoritmo es averiguar las páginas base que son buenos nodos y buenas autoridades y devolver las mejores autoridades y nodos como respuestas de la consulta. Para determinar la calidad de las páginas base como nodos o autoridades se asocia con cada una de ellas un **peso de nodo** y un **peso de autoridad**. El peso de nodo de cada página indica su calidad como nodo, y su peso de autoridad, su calidad como autoridad. Los pesos de cada página se calculan de acuerdo con la intuición de que una página es una buena autoridad si muchos buenos nodos tienen hipervínculos con ella y un buen nodo si tiene muchos hipervínculos salientes con buenas autoridades. Dado que no se tiene un conocimiento previo sobre las páginas que son buenos nodos o autoridades, todos los pesos tienen un valor inicial de uno. Luego, los pesos de autoridad y de nodo de las páginas base se actualizan de manera iterativa tal y como se describe a continuación.

Considérese la página base p con peso de nodo n_p y peso de autoridad a_p . en una iteración se actualiza a_p a la suma de los pesos de nodo de todas las páginas que tienen un hipervínculo con p . Formalmente:

$$a_p = \sum_{\text{Todas las páginas base } q \text{ que tienen un vínculo con } p} n_q$$

De manera análoga, se actualiza n_p a la suma de los pesos de todas las páginas a las que apunta p :

$$n_p = \sum_{\text{Todas las páginas base } q \text{ tales que } p \text{ tiene un enlace con } q} a_q$$

SQL/MM: Full Text (texto completo). El “texto completo” se describe como datos que se pueden buscar, a diferencia de las meras cadenas de caracteres, y se ha introducido un nuevo tipo de datos denominado **FullText** para soportarlo. Los métodos asociados con este tipo soportan la búsqueda de palabras aisladas, expresiones, palabras que “sueñan como” los términos de la búsqueda, etc. Tres métodos resultan de especial interés. **CONTAINS** comprueba si un objeto de FullText contiene un término de búsqueda concreto (palabra o expresión). **RANK** devuelve la clasificación según la relevancia del objeto de FullText con respecto al término de búsqueda especificado. (La manera de definir la clasificación se deja a la implementación concreta.) **IS ABOUT** determina si el objeto de FullText está suficientemente relacionado con el término de búsqueda especificado. (El comportamiento de **IS ABOUT** también se deja a la implementación concreta.) Los SGBD de IBM, Microsoft y Oracle soportan los campos de texto, aunque actualmente no se atienden a la norma de SQL/MM.

Comparando el algoritmo con los otros enfoques de las consultas de texto que se han analizado en este capítulo, se nota que el paso de iteración del algoritmo HITS —la distribución de pesos— no toma en cuenta las palabras de las páginas base. En el paso de iteración sólo nos preocupa la relación entre las páginas base que representan los hipervínculos.

El algoritmo HITS suele ofrecer muy buenos resultados. Por ejemplo, los cinco primeros resultados de Google (que emplea una variante del algoritmo HITS) para la consulta “Raghu Ramakrishnan” son las páginas Web siguientes:

```
www.cs.wisc.edu/~raghu/raghu.html
www.cs.wisc.edu/~dbbook/dbbook.html
www.informatik.uni-trier.de/
    ~ley/db/indices/a-tree/r/Ramakrishnan:Raghu.html
www.informatik.uni-trier.de/
    ~ley/db/indices/a-tree/s/Seshadri:Praveen.html
www.acm.org/awards/fellows_citations_n-z/ramakrishnan.html
```

El primer resultado es la página Web de Ramakrishnan; el segundo es la página Web de este libro; el tercero es la página que refleja sus publicaciones en la popular bibliografía DBLP; y el cuarto resultado (inicialmente sorprendente) es la lista de publicaciones de uno de sus antiguos alumnos.

18.5 GESTIÓN DEL TEXTO EN SGBD

En los apartados anteriores se ha visto el modo en que se indexan y se consultan los grandes conjuntos de texto en los sistemas de RI y en los motores de búsqueda Web. Ahora se considerarán los retos adicionales suscitados por la integración de datos de texto en los sistemas de bases de datos.

El enfoque básico seguido por la comunidad que elabora las normas de SQL es tratar los documentos de texto como un nuevo tipo de datos, **FullText**, que puede aparecer como valor de los campos de las tablas. Si se define una tabla con una sola columna de tipo FullText,

cada fila de esa tabla se corresponde con un documento de un conjunto de documentos. Los métodos de FullText se pueden emplear en la cláusula `WHERE` de las consultas de SQL para recuperar filas que contengan objetos de texto que coincidan según un criterio de búsqueda al estilo RI. La clasificación de relevancia de un objeto de FullText se puede recuperar de manera explícita mediante el método `RANK`, y éste se puede emplear para ordenar los resultados por su relevancia.

Es necesario tener presentes varios aspectos al considerar este enfoque:

- Se trata de un enfoque tremadamente general, y es probable que el rendimiento de los sistemas de SQL que soportan estas extensiones sea inferior al de los sistemas especializados de RI.
- El modelo de datos no refleja adecuadamente los documentos con metadatos adicionales. Si se guardan documentos en tablas con columnas de FullText y se emplean otras columnas para guardar metadatos —por ejemplo, autor, título, resumen, valoración, popularidad— hay que expresar las medidas de la relevancia que combinan los metadatos con las medidas de similitud de RI mediante nuevos métodos definidos por los usuarios, ya que el método `RANK` sólo tiene acceso a los objetos FullText y no a los metadatos. La aparición de los documentos de XML, que tienen metadatos parciales no uniformes, complica aún más las cosas.
- El tratamiento de las actualizaciones no está claro. Como se ha visto, los índices para RI son complejos y costosos de mantener. Exigir que un sistema actualice los índices antes de que la transacción de actualización se comprometa puede imponer una grave penalización al rendimiento.

18.5.1 Índices invertidos débilmente acoplados

El enfoque de implementación empleado en los SGBD actuales que soportan los campos de texto es tener un motor para búsqueda de texto independiente que esté débilmente acoplado con el SGBD. El motor actualiza periódicamente los índices, pero no ofrece ninguna garantía transaccional. Por tanto, una transacción puede insertar (una fila que contenga) un objeto de texto y comprometerse, y puede que una transacción posterior que formule una consulta que coincida no recupere (la fila que contiene) ese objeto.

18.6 MODELO DE DATOS PARA XML

Como se ha visto en el Apartado 7.4.1, XML ofrece una manera de marcar los documentos con etiquetas significativas que les aportan una estructura parcial. Los *modelos de datos semiestructurados*, que se introducen en este apartado, capturan gran parte de la estructura de los documentos de XML, al tiempo que abstraen muchos detalles¹. Los modelos de datos semiestructurados tienen el potencial de servir como base formal para XML y permitir definir rigurosamente la semántica de las consultas XML, que se analizan en el Apartado 18.7.

¹Un aspecto importante de XML que *no* se captura es la ordenación de los elementos. El comité del W3C que desarrolla las normas para XML ha propuesto un modelo de datos más complejo, pero no se analizará aquí.

Modelos de datos para XML. Los comités de normas como ISO y W3C están considerando varios modelos de datos para XML. **Infoset** del W3C es un modelo con estructura de árbol, y cada nodo se puede recuperar mediante una **función de acceso**. Una versión denominada **PSVI** (Post-Validation Infoset) sirve como modelo de datos para XML Schema. El lenguaje XQuery tiene otro modelo de datos asociado. Esta pléthora de modelos se debe en algunos casos a desarrollos en paralelo y en otros a la diversidad de objetivos. No obstante, todos estos modelos tienen árboles poco estructurados como característica principal.

18.6.1 Razón de la estructura laxa

Considérese un conjunto de documentos Web que contengan hipervínculos a otros documentos. Esos documentos, aunque no estén completamente sin estructurar, no se pueden modelar de manera natural en el modelo de datos relacional, ya que la estructura de hipervínculos no es regular de unos documentos a otros. De hecho, cada documento HTML tiene una estructura mínima, como el texto de la etiqueta **TITLE** en contraposición al cuerpo del documento, o el texto destacado frente al que no lo está. Como ejemplo adicional, los archivos de bibliografía también tienen cierto grado de estructura debido a campos como *autor* y *título*, pero es texto sin estructura en todos los demás aspectos. Incluso los datos que “no tienen estructura”, como el texto libre, las imágenes o los fragmentos de vídeo suelen tener alguna información asociada como las marcas de tiempo o la información sobre el autor que aporta una estructura parcial.

Nos referimos a datos con estructuras parciales como la de los **datos semiestructurados**. Hay muchas razones por las cuales los datos pueden estar semiestructurados. En primer lugar, puede que la estructura de los datos esté implícita, oculta, sea desconocida o el usuario decida ignorarla. En segundo lugar, al integrar datos de varios orígenes heterogéneos, el intercambio de datos y su transformación son problemas importantes. Se necesita un modelo de datos muy flexible que integre datos de todos los tipos de orígenes de datos, incluidos los archivos planos y los sistemas heredados; los modelos estructurados como el relacional suelen ser demasiado rígidos. En tercer lugar, no se pueden consultar las bases de datos estructuradas sin conocer su esquema, pero a veces se desea consultar datos sin un conocimiento completo del mismo. Por ejemplo, no se puede expresar la consulta “¿En qué parte de la base de datos se puede hallar la cadena de caracteres *Madrid*? ” en un sistema de bases de datos relacionales sin conocer el esquema y sabiendo los campos que contienen esos valores de texto.

18.6.2 Modelo de grafos

Todos los modelos de datos propuestos para los datos semiestructurados representan los datos como algún tipo de grafo etiquetado. Los nodos del grafo corresponden a objetos compuestos o a valores atómicos. Cada arco indica una relación objeto-subobjeto u objeto-valor. Los nodos hoja, es decir, los nodos sin arcos salientes, tienen un valor asociado con ellos. No hay esquema aparte ni descripción auxiliar; los datos del grafo se describen por sí mismos. Por ejemplo, considérese el grafo de la Figura 18.7, que representa parte de los datos XML de la

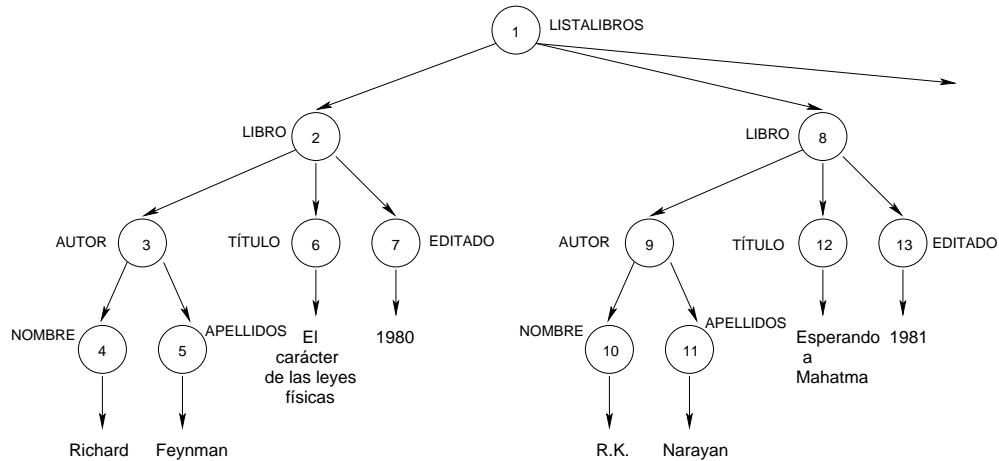


Figura 18.7 Modelo de datos semiestructurados

Figura 7.2. El nodo raíz del grafo representa al elemento más externo, LISTALIBROS. El nodo tiene tres hijos que se etiquetan con el nombre de elemento LIBRO, ya que la lista de libros consiste en tres libros diferentes. Los números de los nodos indican el identificador de objeto asociado con el objeto correspondiente.

A continuación se describirá uno de los modelos de datos propuestos para los datos semiestructurados, denominado **modelo de intercambio de objetos** (object exchange model, OEM). Cada objeto se describe mediante una cuádrupla consistente en una *etiqueta*, un *tipo*, el *valor* del objeto y un *identificador del objeto*, que es un identificador único para cada objeto. Dado que cada objeto tiene una etiqueta que se puede considerar como el nombre de una columna del modelo relacional, y un tipo que se puede considerar como el tipo de columna del modelo relacional, el modelo de intercambio de objetos se describe a sí mismo. Las etiquetas del OEM deben ser todo lo informativas que sea posible, ya que cumplen dos propósitos: se pueden utilizar para identificar los objetos y para transmitir su significado. Por ejemplo, se pueden representar los apellidos de un autor de la manera siguiente:

`<Apellidos, string, "Feynman">`

Los objetos más complejos se descomponen jerárquicamente en objetos más pequeños. Por ejemplo, el nombre de un autor puede contener nombre y apellidos. Este objeto se describe de la manera siguiente:

```

<nombreAutor, set, {nombre1, apellidos1}>
  nombre1 is <Nombre, string, "Richard">
  apellidos1 is <Apellidos, string, "Feynman">
  
```

Como ejemplo adicional, los objetos que representan conjuntos de libros se describen de la manera siguiente:

```

<listaLibros, set, {libro1, libro2, libro3}>
  libro1 is <libro, set, {autor1, título1, editado1}>
  
```

SQL y XML. XQuery es una norma propuesta por el Consorcio World Wide Web (World-Wide Web Consortium, W3C). En paralelo, los comités de normas que desarrollan las normas del SQL han estado trabajando en un sucesor para SQL:1999 que soporte XML. La parte que se refiere a XML se ha denominado, en principio, **SQL/XML** y se pueden encontrar detalles sobre la misma en <http://sqlx.org>.

```

libro2 is <libro, set, {autor2, título2, editado2}>
libro3 is <libro, set, {autor3, título3, editado3}>
autor3 is <autor, set, {nombre3, apellidos3}>
título3 is <título, string, "El profesor de inglés">
editado3 is <editado, integer, 1980>
```

18.7 XQUERY: CONSULTA DE DATOS XML

Dado que los documentos XML se codifican de manera que refleja (un considerable grado de) estructura, aparece la oportunidad de utilizar lenguajes de alto nivel que aprovechen esa estructura para recuperar los datos de esos documentos de la manera más conveniente. Esos lenguajes también permiten trasladar los datos XML entre diferentes DTD, como se debe hacer al integrar datos de varios orígenes. En el momento de escribir este libro **XQuery** es el lenguaje de consulta de la norma del W3C para los datos XML. En este apartado se ofrece una breve visión general de XQuery.

18.7.1 Expresiones de ruta

Considérese el documento XML de la Figura 7.2. La consulta de ejemplo siguiente devuelve los apellidos de todos los autores, suponiendo que el documento XML reside en la ubicación www.nuestralibreria.com/libros.xml.

```

FOR
  $a IN doc(www.nuestralibreria.com/libros.xml)//AUTOR/APELLIDOS
RETURN <RESULT> $a </RESULT>
```

Este ejemplo ilustra algunas de las estructuras básicas de XQuery. La cláusula **FOR** de XQuery es básicamente análoga a la cláusula **FROM** del SQL. La cláusula **RETURN** es parecida a la cláusula **SELECT**. En breve volveremos a la forma general de las consultas, una vez introducido un concepto importante denominado **expresión de ruta**.

La expresión

```
doc(www.nuestralibreria.com/libros.xml)//AUTOR/APELLIDOS
```

de la cláusula **FOR** es un ejemplo de expresión de ruta. Especifica una **ruta** que implica a tres entidades: el propio documento, los elementos de **AUTOR** y los elementos de **APELLIDOS**.

La relación de ruta se expresa mediante los separadores **/** y **//**. El separador **//** especifica que el elemento **AUTOR** puede estar anidado en cualquier parte del documento, mientras que el separador **/** restringe al elemento **APELLIDOS** a estar anidado inmediatamente bajo (en

XPath y otros lenguajes de consulta de XML. Las expresiones de ruta de XQuery se derivan de XPath, una utilidad de consulta de XML anterior. Las expresiones de ruta de XPath se pueden modificar con condiciones de selección y pueden emplear varias funciones intrínsecas (por ejemplo, contar el número de nodos que coinciden con la expresión). Muchas de las características de XQuery proceden de lenguajes anteriores, incluidos XML-QL y Quilt.

términos de la estructura de grafo del documento) el elemento AUTOR. La evaluación de las expresiones de ruta devuelve un *conjunto* de elementos que coinciden con la expresión. La variable *a* de la consulta de ejemplo se halla ligada, a su vez, a cada elemento de APELLIDOS devuelto al evaluar la expresión de ruta. (Para distinguir los nombres de variable del texto normal, en XQuery los nombres de variable van precedidos por un signo de dólar \$.)

La cláusula RETURN crea el resultado de la consulta —que también es un documento XML— escribiendo entre paréntesis cada valor al que esté ligada la variable *a* con la etiqueta RESULT. Si se aplica la consulta de ejemplo al ejemplar de datos de la Figura 7.2, el resultado será el siguiente documento XML:

```
<RESULT><APELLIDOS>Feynman </APELLIDOS></RESULT>
<RESULT><APELLIDOS>Narayan </APELLIDOS></RESULT>
```

El documento de la Figura 7.2 se utiliza como entrada para el resto del capítulo.

18.7.2 Expresiones FLWR

La forma básica de XQuery consiste en una **expresión FLWR**, donde las letras denotan las cláusulas FOR, LET, WHERE y RETURN. Las cláusulas FOR y LET vinculan las variables con valores mediante expresiones de ruta. Esos valores se modifican mediante la cláusula WHERE, y el fragmento de XML resultante se construye mediante la cláusula RETURN.

La diferencia entre las cláusulas FOR y LET radica en que, mientras que con la cláusula FOR se vincula la variable a cada elemento especificado por la expresión de ruta, con la cláusula LET se vincula a todo el *conjunto* de elementos. Por tanto, si se modifica la consulta de ejemplo hasta que quede como:

```
LET
$a IN doc(www.nuestralibreria.com/libros.xml)//AUTOR/APELLIDOS
RETURN <RESULT> $a </RESULT>
```

el resultado de la consulta pasa a ser:

```
<RESULT>
  <APELLIDOS>Feynman</APELLIDOS>
  <APELLIDOS>Narayan</APELLIDOS>
</RESULT>
```

Las condiciones de selección se expresan mediante la cláusula WHERE. Además, el resultado de cada consulta no está limitado a un solo elemento. Estos aspectos se ilustran mediante la

consulta siguiente, que halla el nombre y los apellidos de todos los autores que han escrito libros editados en 1980:

```
FOR $1 IN doc(www.nuestralibreria.com/libros.xml)/BOOKLIST/BOOK
WHERE $1/EDITADO='1980'
RETURN
<RESULT> $1/AUTOR/NOMBRE, $1/AUTOR/APELLIDOS </RESULT>
```

El resultado de la consulta anterior es el siguiente documento XML:

```
<RESULT>
<NOMBRE>Richard </NOMBRE><APELLIDOS>Feynman </APELLIDOS>
</RESULT>
<RESULT>
<NOMBRE>R.K. </NOMBRE><APELLIDOS>Narayan </APELLIDOS>
</RESULT>
```

Para el DTD de este ejemplo, donde cada elemento de LIBRO sólo tiene un AUTOR, la consulta anterior se puede escribir empleando una expresión de ruta diferente para la cláusula FOR, de la manera siguiente.

```
FOR $e IN
doc(www.nuestralibreria.com/libros.xml)
/LISTALIBROS/LIBRO[EDITADO='1980']/AUTOR
RETURN <RESULT> $e/NOMBRE, $e/APELLIDOS </RESULT>
```

La expresión de ruta de esta consulta es un ejemplo de **expresión de ruta con bifurcaciones**. La variable *e* se vincula ahora a cada elemento de AUTOR que coincida con la ruta doc/LISTALIBROS/LIBRO/AUTOR, en la que el elemento intermedio LIBRO está restringido a tener un elemento EDITADO anidado inmediatamente en su interior con el valor 1980.

18.7.3 Ordenación de los elementos

Los datos XML consisten en documentos *ordenados* y, por tanto, el lenguaje de consultas debe devolver los datos en un cierto orden. La semántica de XQuery es que cada expresión de ruta devuelva los resultados ordenados según el orden de los documentos. Así, las variables de la cláusula FOR se vinculan según el orden de los documentos. Si, no obstante, se desea un orden diferente, se puede aplicar al resultado de manera explícita, como puede verse en la consulta siguiente, que devuelve los elementos de TITULO ordenados lexicográficamente.

```
FOR
$1 IN doc(www.nuestralibreria.com/libros.xml)/LISTALIBROS/LIBRO
RETURN <TI TULOSLIBROS> $1/TITULO </TI TULOSLIBROS>
SORT BY TITULO
```

18.7.4 Agrupación y generación de colecciones

El ejemplo siguiente ilustra la agrupación en XQuery, que permite generar nuevos valores de tipo colección para cada grupo. (Compárese esto con la agrupación en SQL, que sólo permite

generar valores agregados [por ejemplo, SUM] para cada grupo.) Supóngase que para cada año se desea averiguar los apellidos de los autores que han escrito libros editados ese año. Agrupamos por año de edición y se genera una lista de apellidos para cada año:

```
FOR $p IN DISTINCT
  doc(www.nuestralibreria.com/libros.xml)/LISTALIBROS/LIBRO/EDITADOS
  RETURN
<RESULT>
  $p,
  FOR $e IN DISTINCT /LISTALIBROS/LIBRO[EDITADO=$e]/AUTOR
    RETURN $e
</RESULT>
```

La palabra clave *DISTINCT* elimina los valores duplicados del conjunto devuelto por la expresión de ruta. Con el documento XML de la Figura 7.2 como entrada, la consulta anterior produce el resultado siguiente:

```
<RESULT> <EDITADO>1980</EDITADO>
  <APELIDOS>Feynman</APELIDOS>
  <APELIDOS>Narayan</APELIDOS>
</RESULT>
<RESULT> <EDITADO>1981</EDITADO>
  <APELIDOS>Narayan</APELIDOS>
</RESULT>
```

18.8 EVALUACIÓN EFICIENTE DE CONSULTAS XML

XQuery opera con los datos XML y produce datos XML como resultado. Para poder evaluar eficientemente las consultas hace falta abordar los problemas siguientes.

- **Almacenamiento.** Se pueden utilizar sistemas de almacenamiento existentes como los sistemas relacionales o los orientados a objetos o diseñar un nuevo formato de almacenamiento para los documentos de XML. Hay varias maneras de utilizar los sistemas relacionales para almacenar XML. Uno de ellos es almacenar los datos del XML como objetos de caracteres de gran tamaño (Character Large Objects, CLOB). (Los CLOB se examinaron en el Capítulo 15.) En este caso, sin embargo, no se puede aprovechar la infraestructura de procesamiento de consultas proporcionada por el sistema relacional y, en su lugar, hay que procesar XQuery fuera del sistema de bases de datos. Para evitar este problema hace falta identificar un esquema según el cual se puedan almacenar los datos XML. Estos puntos se analizan en el Apartado 18.8.1.
- **Indexación.** Las expresiones de ruta añaden mucha riqueza a XQuery y ofrecen muchas pautas de acceso a los datos nuevas. Si se utiliza un sistema relacional para almacenar los datos XML, sólo se pueden utilizar índices relacionales como el árbol B. Sin embargo, si se utilizan motores nativos de almacenamiento, se tiene la posibilidad de crear nuevas estructuras de índice para las expresiones de ruta, algunas de las cuales se analizan en el Apartado 18.8.2.

Los sistemas comerciales de bases de datos y XML. Actualmente muchos fabricantes de sistemas de bases de datos orientadas a objetos intentan dar soporte a XML en sus motores de bases de datos. Varios fabricantes de sistemas de gestión de bases de datos orientadas a objetos ya ofrecen motores de bases de datos que pueden almacenar datos XML a cuyo contenido se puede tener acceso mediante interfaces gráficas de usuario o extensiones de Java en el lado del servidor.

- **Optimización de consultas.** La optimización de consultas en XQuery es un problema abierto. Hasta ahora el trabajo en esta área se puede dividir en tres partes. La primera es el desarrollo de un álgebra para XQuery, análogo al álgebra relacional. La segunda dirección de la investigación es conseguir estadísticas de las consultas de expresiones de ruta. Finalmente, algunos trabajos han abordado la simplificación de las rutas mediante el aprovechamiento de restricciones a los datos. Dado que la optimización de las consultas para XQuery sigue hallándose en una etapa preliminar, no se tratará en este capítulo.

Otro aspecto que se debe considerar al diseñar nuevos sistemas de almacenamiento para los datos XML es la verbosidad de las etiquetas repetidas. Como puede verse en el Apartado 18.8.1, el empleo de sistemas de almacenamiento relacionales aborda este problema, ya que los nombres de las etiquetas no se almacenan de manera repetida. Si, por otro lado, se desea crear un sistema nativo de almacenamiento, el modo en que se compriman los datos XML pasa a ser significativo. Se conocen varios algoritmos de compresión que consiguen tasas de compresión cercanas a las del almacenamiento relacional, pero no se estudiarán aquí.

18.8.1 Almacenamiento de XML en SGBDR

Un candidato natural para el almacenaje de datos XML son los sistemas de bases de datos relacionales. Los principales problemas relacionados con el almacenamiento de datos XML en sistemas relacionales son:

- *Elección del esquema relacional.* Para utilizar un SGBDR hace falta un esquema. ¿Qué esquema relacional se debe utilizar aunque se suponga que los datos XML vienen con un esquema asociado?
- *Consultas.* Las consultas de datos XML se realizan en XQuery, mientras que los sistemas relacionales sólo pueden manejar SQL. Por tanto, hace falta *traducir* las consultas de XQuery a SQL.
- *Reconstrucción.* El resultado de XQuery es XML. Por tanto, hace falta volver a convertir en XML el resultado de las consultas de SQL.

Correspondencia entre los datos XML y las relaciones

Se ilustrará el proceso de correspondencia mediante el ejemplo de la librería. Las relaciones que se anidan entre los diferentes elementos de DTD pueden verse en la Figura 18.8. Los arcos indican la naturaleza del anidamiento.

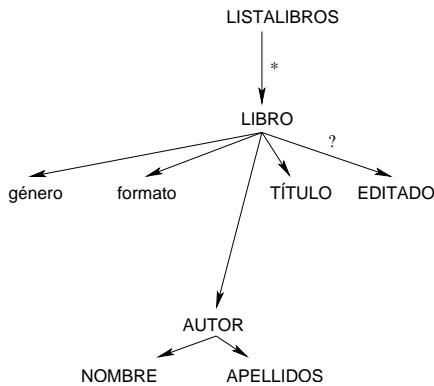


Figura 18.8 Relaciones entre los elementos de DTD de XML de la librería

Una manera de obtener esquemas relacionales es la siguiente. Se comienza con el elemento **LISTALIBROS** y se crea una relación que lo almacene. Bajando desde **LISTALIBROS**, se llega a **LIBRO** siguiendo una arco *. Ese arco indica que los elementos de **BOOK** se almacenan en una relación diferente. Bajando más, se ve que todos los elementos y atributos anidados en **LIBRO** aparecen, como máximo, una vez. Por tanto, se pueden almacenar en la misma relación que **LIBRO**. El esquema relacional resultante *EsquemaRel1* puede verse más abajo.

```

LISTALIBROS(id: integer)
LIBRO (idlistalibros: integer, nombre_autor: string,
        apellidos_autor: string, título: string,
        editado: string, género: string, formato: string)
  
```

La columna **LIBRO.idlistalibros** conecta **LIBRO** con **LISTALIBROS**. Dado que cada DTD sólo tiene un tipo básico, **string**, el único tipo básico utilizado en el esquema anterior es **string**. Las restricciones expresadas mediante el DTD se expresan en el esquema relacional. Por ejemplo, dado que todo **LIBRO** debe tener un **TÍTULO** hijo, se debe restringir la columna **título** a que no sea *null*.

De manera alternativa, si se modifica el DTD para permitir que **LIBRO** tenga más de un **AUTOR** hijo, no se pueden almacenar los elementos de **AUTOR** en la misma relación que **LIBRO**. Esta modificación da como resultado el siguiente esquema relacional, *EsquemaRel2*.

```

LISTALIBROS(id: integer)
LIBRO (id: integer, idlistalibros: integer,
        título: string, editado: string, género: string, formato: string)
AUTOR(idlibro: integer, nombre: string, apellidos: string)
  
```

La columna **AUTOR.idlibro** conecta **AUTOR** con **LIBRO**.

Procesamiento de consultas

Considérese de nuevo la siguiente consulta de ejemplo:

FOR

```
$1 IN doc(www.nuestralibreria.com/libros.xml)/LISTALIBROS/LIBRO
WHERE $1/EDITADO='1980'
RETURN
<RESULT> $1/AUTOR/NOMBRE, $1/AUTOR/APELLIDOS </RESULT>
```

Si se conoce la correspondencia entre los datos XML y las tablas relacionales, se puede crear una consulta SQL que devuelva todas las columnas necesarias para reconstruir el documento XML resultante para esta consulta. Las condiciones aplicadas por las expresiones de ruta y la cláusula WHERE se traducen en las condiciones equivalentes de la consulta SQL. Se obtiene la siguiente consulta SQL equivalente si se utiliza *EsquemaRel1* como esquema relacional.

```
SELECT LIBRO.nombre_autor, LIBRO.apellidos_autor
FROM LIBRO, LISTALIBROS
WHERE LISTALIBROS.id = LIBRO.idlistalibros
AND LIBRO.editado='1980'
```

Los resultados devueltos por el procesador de la consulta relacional se etiquetan, fuera del sistema relacional, tal y como especifica la cláusula RETURN. Éste es el resultado de la fase de *reconstrucción*.

Para comprenderlo mejor, considérese lo que sucede si se permite que LIBRO tenga varios hijos AUTOR. Supóngase que se utiliza *EsquemaRel2* como esquema relacional. El procesamiento de las cláusulas FOR y WHERE indica que es necesario reunir las relaciones LISTALIBROS y LIBRO con una selección sobre la relación LIBRO que corresponda a la condición de año de la consulta anterior. Dado que la cláusula RETURN necesita información sobre los elementos de AUTOR, hay que reunir nuevamente la relación LIBRO con la relación AUTOR y proyectar las columnas *nombre* y *apellidos* en esta última. Finalmente, como cada vínculo de la variable \$1 de la consulta anterior produce un elemento de RESULTADO y, dado que ahora se permite que cada elemento de BOOK tenga más de un AUTOR, hay que proyectar la columna *id* de la relación LIBRO. Con base en estas observaciones se obtiene la siguiente consulta equivalente de SQL:

```
SELECT LIBRO.id, AUTOR.nombre, AUTOR.apellidos
FROM LIBRO, LISTALIBROS, AUTOR
WHERE LISTALIBROS.id = LIBRO.idlistalibros AND
LIBRO.id = AUTOR.idlibro AND LIBRO.editado='1980'
GROUP BY LIBRO.id
```

El resultado se agrupa según LIBRO.id. El etiquetador externo al sistema de bases de datos recibe ahora los resultados agrupados por el elemento de LIBRO y puede etiquetar las tuplas resultantes sobre la marcha.

Publicación de datos relacionales como XML

Desde que XML ha emergido como formato estándar para el intercambio de datos para las aplicaciones empresariales es necesario publicar los datos empresariales existentes como XML. La mayor parte de los datos de operaciones comerciales se almacenan en sistemas relacionales. En consecuencia, se han propuesto mecanismos para editar esos datos como documentos XML.

Esos mecanismos incorporan un lenguaje para la especificación del modo en que se deben etiquetar y estructurar los datos relacionales y una implementación para llevar a cabo la conversión. Esta correspondencia es, en cierto sentido, la inversa de la correspondencia XML-a-relacional empleada para almacenar los datos XML. El proceso de conversión reproduce la fase de reconstrucción de la ejecución de XQuery mediante un sistema relacional. Los datos XML editados se pueden considerar como una vista XML de los datos relacionales. Esa vista se puede consultar mediante XQuery. Un método de ejecución de XQuery en esas vistas es traducirlas a SQL y crear luego el resultado en XML.

18.8.2 Indexación de repositorios XML

Las expresiones de ruta se hallan en el corazón de todos los lenguajes de consulta de XML propuestos, en especial de XQuery. Una pregunta que surge de manera natural es cómo indexar los datos XML para soportar la evaluación de las expresiones de ruta. El objetivo de este apartado es ofrecer una pequeña introducción a las técnicas de indexación propuestas para este problema. Se considerará el modelo OEM de datos semiestructurados, en el que los datos se describen a sí mismos y no hay un esquema independiente.

Empleo de árboles B+ para indexar valores

Considérese el siguiente ejemplo de XQuery, que ya se analizó para los datos XML de la librería en la Figura 7.2. La representación OEM de estos datos puede verse en la Figura 18.7.

```

FOR
  $l IN doc(www.nuestralibreria.com/libros.xml)/LISTALIBROS/LIBRO
  WHERE $l/EDITADO='1980'
RETURN
<RESULT> $l/AUTOR/NOMBRE, $l/AUTOR/APELLIDOS </RESULT>

```

Esta consulta especifica reuniones entre los objetos con las etiquetas LISTALIBROS, LIBRO, AUTOR, NOMBRE, APELLIDOS y EDITADO con una condición de selección para los objetos EDITADO.

Supongamos que se evalúa esta consulta en ausencia de índices para las expresiones de ruta. Sin embargo, se tiene un índice de valores como el árbol B que permite averiguar los identificadores de todos los objetos con la etiqueta EDITADO y el valor 1980. Hay varias maneras de ejecutar esta consulta con estas suposiciones.

Por ejemplo, se podría empezar en la raíz del documento y descender por el grafo de datos por el objeto LISTALIBROS hasta los objetos LIBRO. Si se sigue descendiendo por el gráfico de datos, se puede comprobar si cada objeto LIBRO satisface el predicado de valores (`EDITADO='1980'`). Finalmente, para los objetos LIBRO que satisfacen el predicado, se pueden hallar los objetos NOMBRE y APELLIDOS relevantes. Este enfoque corresponde a una evaluación descendente de la consulta.

También se podría comenzar utilizando el índice de valores para buscar todos los objetos EDITADO que satisfacen `EDITADO='1980'`. Si se puede recorrer el grafo de datos en sentido contrario —es decir, dado un objeto, se puede averiguar su padre— se pueden hallar todos los padres de los objetos EDITADO y conservar sólo los que tengan la etiqueta LIBRO. Se puede

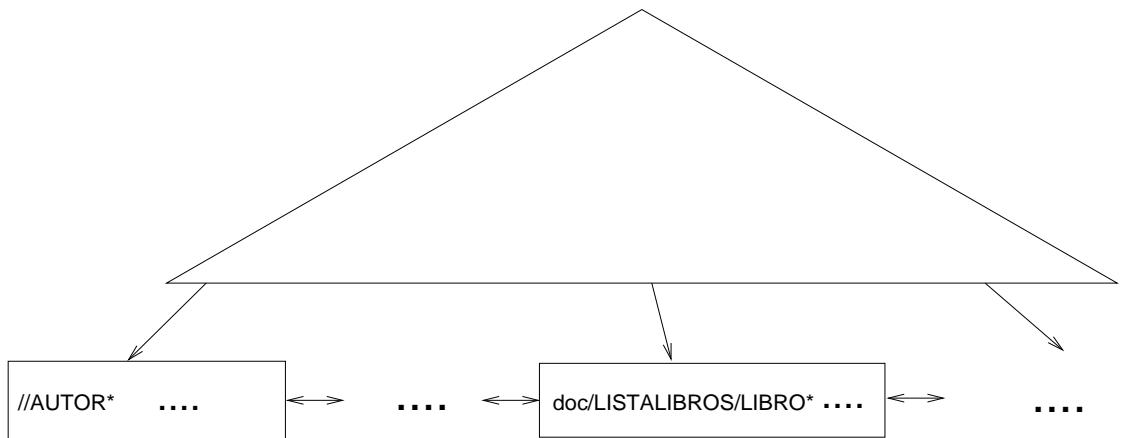


Figura 18.9 Expresiones de ruta en un árbol B

continuar de esta manera hasta hallar los objetos NOMBRE y APELLIDOS de interés. Obsérvese que hay que llevar a cabo sobre la marcha todas las reuniones de la consulta.

Indexación de estructuras y de valores

Preguntémonos ahora si las soluciones tradicionales de indexación como los árboles B se pueden utilizar para indexar las expresiones de ruta. Se pueden utilizar los árboles B para asignar expresiones de ruta a los identificadores de todos los objetos devueltos por esas expresiones. La idea es tratar todas las expresiones de ruta como cadenas de caracteres y ordenarlas lexicográficamente. Cada hoja del árbol B contiene una cadena de caracteres que representa una expresión de ruta y una lista de los identificadores correspondientes a su resultado. La Figura 18.9 muestra el aspecto que tendría tal árbol B. Compárese esto con el problema tradicional de la indexación de dominios bien ordenados como los enteros de las consultas de puntos. En este último caso, el número de consultas de puntos diferentes que se pueden plantear es exactamente el número de valores de datos y, por tanto, es lineal con el tamaño de los datos.

La situación con la indexación de rutas es fundamentalmente diferente —la variedad de modos en que se pueden combinar las etiquetas para formar expresiones de ruta (sencillas) junto con la posibilidad de situar separadores // lleva a un número de expresiones de ruta muy superior—. Por ejemplo, se devuelve un elemento AUTOR del ejemplo de la Figura 18.7 como parte de las consultas LISTALIBROS/LIBRO/AUTOR, //AUTOR, //LIBRO//AUTOR, LISTALIBROS//AUTOR, etcétera. El número de consultas diferentes puede, de hecho, ser exponencial en el tamaño de los datos (medidos en términos del número de elementos XML) en el peor de los casos. Esto es lo que motiva la búsqueda de estrategias alternativas para la indexación de las expresiones de ruta.

El enfoque adoptado es representar la correspondencia entre cada expresión de ruta y su resultado mediante un resumen estructural que toma la forma de otro grafo etiquetado y dirigido. La idea es conservar todas las rutas del grafo de datos en el grafo resumen, al

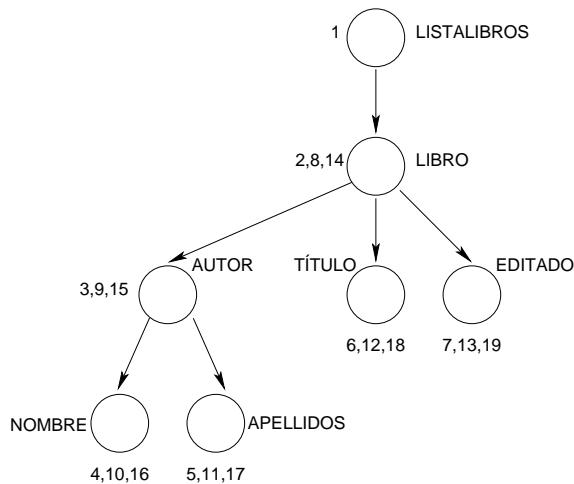


Figura 18.10 Ejemplo de índice de rutas

tiempo que se tienen muchos menos nodos y arcos. Se asocia una **extensión** con cada nodo del resumen. La extensión del nodo de un índice es un subconjunto de los nodos de datos. El grafo resumen constituye, junto con las extensiones, un índice de rutas. Las expresiones de ruta se evalúan mediante el índice evaluándolo frente al grafo resumen y tomando la unión de las extensiones de todos los nodos coincidentes. Esto da el resultado de índices de la consulta de expresiones de ruta. El índice **cubre** una expresión de ruta si el índice resultante es el resultado correcto; evidentemente, sólo se puede utilizar un índice para evaluar una expresión de ruta si el índice la cubre.

Considérese el resumen estructural que aparece en la Figura 18.10. Se trata de un índice de rutas para los datos de la Figura 18.7. Los números que aparecen al lado de los nodos corresponden a las extensiones respectivas. Examinemos ahora la manera en que este índice puede cambiar la evaluación de descendente de la consulta de ejemplo empleada anteriormente para ilustrar los índices de valores de los árboles B+.

La evaluación descendente tal y como se ha esbozado antes comienza en la raíz del documento y desciende hasta los objetos LIBRO. Esto se puede conseguir de manera más eficiente mediante el índice de rutas. En lugar de recorrer el grafo de datos, se puede recorrer el índice de rutas hasta el objeto LIBRO del índice y buscar su extensión, lo que proporciona los identificadores de todos los objetos LIBRO que coinciden con la expresión de ruta de la cláusula FOR. El resto de la evaluación sigue igual que antes. Por tanto, el índice de rutas evita llevar a cabo reuniones, básicamente, calculándolas previamente. Aquí hay que destacar que el índice de rutas de la Figura 18.10 es isomorfo con el grafo de esquemas DTD de la Figura 18.8. Esto confirma el hecho de que el índice de rutas sin las extensiones es un resumen estructural de los datos.

El índice de rutas anterior es la **guía fuerte de datos**. Si se tratan las expresiones de ruta como cadenas de caracteres, la guía de datos es el **árbol de prefijos** que los representa. El árbol de prefijos es una estructura de datos bien conocida empleada para buscar expresiones regulares en el texto. Esto muestra la profunda unidad entre la investigación en indexación de

textos y el trabajo en indexación de rutas en XML. También se han propuesto otros índices de ruta para los datos semiestructurados, que son un área de investigación activa.

18.9 PREGUNTAS DE REPASO

Las respuestas a las preguntas de repaso se pueden encontrar en los apartados indicados.

- ¿Qué es la recuperación de información? (**Apartado 18.1**)
- Indíquense algunas diferencias entre los SGBD y los sistemas de RI. Describanse las diferencias entre una consulta de clasificación y una consulta booleana. (**Apartado 18.2**)
- ¿Qué es el modelo de espacio vectorial y cuáles son sus ventajas? (**Apartado 18.2.1**)
- ¿Qué es el peso FT/FID de los términos y por qué se pesan con ambos? ¿Por qué se eliminan las palabras de parada? ¿Qué es la normalización de longitudes y por qué se lleva a cabo? (**Apartado 18.2.2**)
- ¿Cómo se mide la similitud entre documentos? (**Apartado 18.2.3**)
- ¿Qué son la *precisión* y la *recuperación* y cómo se relacionan entre sí? (**Apartado 18.2.4**)
- Describanse las siguientes estructuras de índice para texto: índice invertido y archivo de firmas. ¿Qué es un archivo de firmas cortado bit a bit? (**Apartado 18.3**)
- ¿Cómo se estructuran los motores de búsqueda Web? ¿Cómo funciona el algoritmo de “nodos y autoridades”? ¿Puede ilustrarlo para un conjunto pequeño de páginas? (**Apartado 18.4**)
- ¿Qué soporte hay para la gestión de texto en los SGBD? (**Apartado 18.5**)
- Describa el modelo de datos OEM para los datos semiestructurados. (**Apartado 18.6**)
- ¿Cuáles son los elementos de XQuery? ¿Qué son las expresiones de ruta? ¿Qué son las expresiones FLWR? ¿Cómo se pueden ordenar los resultados de las consultas? ¿Cómo se agrupan los resultados de las consultas? (**Apartado 18.7**)
- Describase el modo en que los datos XML se pueden almacenar en SGBD relacionales. ¿Cómo se relacionan los datos XML con las relaciones? ¿Se puede utilizar la infraestructura de procesamiento de consultas de los SGBD relacionales? ¿Cómo se publican los datos relacionales como XML? (**Apartado 18.8.1**)
- ¿Cómo se indexan los conjuntos de documentos XML? ¿Cuál es la diferencia entre indexar las estructuras y los valores? ¿Qué es un índice de rutas? (**Apartado 18.8.2**)

EJERCICIOS

Ejercicio 18.1 Llévense a cabo las tareas siguientes.

- Dado un archivo ASCII, calcúlese la frecuencia de cada palabra y dibújese una gráfica parecida a la Figura 18.3. (Se puede utilizar software de dibujo de dominio público.) Ejecute el programa sobre el conjunto de archivos que se hallan actualmente en un directorio y compruébese si la distribución de frecuencias es Zipfiana. ¿Cómo se pueden utilizar esas gráficas para crear listas de palabras de parada?
- Se utiliza mucho el buscador de raíces Porter, y se puede disponer libremente del código que lo implementa. Descárguese una copia y ejecútense sobre un conjunto de documentos.
- Una de las críticas al modelo del espacio vectorial y a su empleo en la comprobación de similitudes es que trata a los términos como si aparecieran independientemente unos de otros. En la práctica, muchas palabras tienden a aparecer juntas (por ejemplo, ambulancia y urgencias). Escríbase un programa que explore un archivo ASCII y relacione todas las parejas de palabras que aparezcan a una distancia máxima de cinco palabras la una de la otra. Ahora se tiene una frecuencia para cada pareja de palabras y se debería poder crear una gráfica como la Figura 18.3 con las parejas de palabras en el eje X. Ejecútense ese programa sobre algunos ejemplos de conjuntos de documentos. ¿Qué sugieren los resultados acerca de las apariciones conjuntas de palabras?

Ejercicio 18.2 Supóngase que se recibe una base de datos documental que contiene seis documentos. Tras buscar las raíces de las palabras, los documentos contienen los términos siguientes:

Documento	Términos
1	coche fabricante Honda automóvil
2	automóvil ordenador navegación
3	Honda navegación
4	fabricante ordenador IBM
5	IBM personal ordenador
6	coche Escarabajo VW

Respóndase a las preguntas siguientes.

- Muéstrese el resultado de la creación de un archivo invertido para los documentos.
- Muéstrese el resultado de la creación de un archivo de firmas con una anchura de 5 bits. Créese su propia función de asociación que haga que se correspondan los términos con las posiciones de los bits.
- Evalúense las siguientes consultas booleanas utilizando el archivo invertido y el archivo de firmas que se acaba de crear: “coche”, “IBM” AND “ordenador”, “IBM” AND “coche”, “IBM” OR “automóvil” e “IBM” AND “ordenador” AND “fabricante”.
- Supóngase que la carga de la consulta sobre la base de datos documental consiste exactamente en las consultas que se han formulado en la pregunta anterior. Supóngase también que cada una de esas consultas se evalúa exactamente una vez.
 - Diséñese un archivo de firmas con una anchura de 3 bits y una función de asociación que minimice el número global de falsos positivos.
 - Diséñese un archivo de firmas con una anchura de 6 bits y una función de asociación que minimice el número global de falsos positivos.
 - Supóngase que se desea crear una archivo de firmas. ¿Cuál es la mínima anchura de firmas que permite evaluar todas las consultas sin recuperar ningún falso positivo?
- Considérense las siguientes consultas de clasificación: “coche”, “ordenador IBM”, “coche IBM”, “automóvil IBM” y “fabricante ordenador IBM”.
 - Calcúlese la FID de cada término de la base de datos.
 - Para cada documento, muéstrese su vector documental.
 - Para cada consulta, calcúlese la relevancia de cada documento de la base de datos, con el paso de normalización de longitudes y sin él.

- (d) Describábase la manera en que se podría utilizar el índice invertido para identificar los dos documentos que más coincidan con cada consulta.
- (e) ¿Cómo afectaría a la respuesta de la pregunta anterior que se ordenaran las listas invertidas por relevancia en lugar de por identificador de documento?
- (f) Sustitúyase cada documento por una variación que contenga diez copias de ese mismo documento. Para cada consulta, vuélvase a calcular la relevancia de cada documento, con el paso de normalización de longitudes y sin él.

Ejercicio 18.3 Supóngase que se recibe la siguiente base de datos documental con las raíces de las palabras ya buscadas:

Documento	Términos
1	coche coche fabricante coche coche Honda automóvil
2	automóvil ordenador navegación
3	Honda navegación automóvil
4	fabricante ordenador IBM gráficos
5	IBM personal IBM ordenador IBM IBM IBM IBM
6	coche Escarabajo VW Honda

Empleando esta base de datos, repítase el ejercicio anterior.

Ejercicio 18.4 Está encargado del motor de búsqueda Gengis (“Ejecutamos más rápido”.) Está diseñando el grupo de servidores para que maneje 500 millones de peticiones diarias y 10.000 millones de páginas de datos indexados. Cada ordenador cuesta 1.000 € y puede almacenar 10 millones de páginas y responder a 200 consultas por segundo (sobre esas páginas).

1. Si se le diera un presupuesto de 500.000 € para comprar ordenadores y se le exigiera que indexara los 10.000 millones de páginas, ¿podría hacerlo?
2. ¿Cuál es el presupuesto mínimo para indexar todas las páginas? Si se supone que cada consulta se puede responder examinando los datos de una sola partición (10 millones de páginas) y que las consultas están distribuidas de manera uniforme por todas las particiones, ¿cuál es la carga pico (en número de consultas por segundo) que puede manejar un grupo de ordenadores como ése?
3. ¿Cómo respondería a la pregunta anterior si cada consulta, en promedio, necesitara tener acceso a dos particiones?
4. ¿Cuál es el presupuesto mínimo necesario para manejar la carga deseada de 500 millones de consultas diarias si todas las consultas se realizan a una *sola partición*? Supóngase que las consultas están distribuidas de manera uniforme a lo largo del día.
5. ¿Cómo cambiaría la respuesta a la pregunta anterior si el número diario de consultas subiera hasta 5.000 millones? ¿Cómo cambiaría si el número de páginas subiera hasta 100.000 millones?
6. Supóngase que cada consulta tiene acceso a una sola partición y que las consultas están distribuidas de manera uniforme por todas las particiones pero que, en cualquier momento, la carga pico de una partición llega a ser de diez veces la carga promedio. ¿Cuál es el presupuesto mínimo para la compra de ordenadores en esta situación?
7. Tome el coste de los ordenadores de la consulta anterior y multiplíquelo por diez para que refleje los costes de mantenimiento, administración, ancho de banda, etcétera. Esta cantidad es el coste anual de funcionamiento. Supóngase que se cobra a los anunciantes 2 céntimos por página. ¿Qué parte del inventario (es decir, del número total de páginas que se facilitan en el transcurso de un año) hay que vender para obtener beneficios?

Ejercicio 18.5 Supóngase que el conjunto base del algoritmo HITS consiste en el conjunto de páginas Web mostradas en la tabla siguiente. Cada entrada se debería interpretar de esta manera: la página Web 1 tiene hipervínculos con las páginas 5, 6 y 7.

Página Web	Páginas con las que esta página tiene vínculos
1	5, 6, 7
2	5, 7
3	6, 8
4	
5	1, 2
6	1, 3
7	1, 2
8	4

1. Ejecute cinco iteraciones del algoritmo HITS y determine la autoridad y el nodo mejor clasificados.
2. Calcúlese el orden de paloma de Google para cada página.

Ejercicio 18.6 Considérese la siguiente descripción de los artículos que aparecen en el catálogo de venta por correo de ordenadores de Carahuevo.

“Carahuevo vende hardware y software. Vendemos la nueva Palm Pilot V por 400 €; su número de artículo es 345. También vendemos el IBM ThinkPad 570 por sólo 1.999 €; su número de artículo es 3.784. Vendemos tanto software de negocios como de entretenimiento. Acabamos de recibir Microsoft Office 2000 y puede comprar la Standard Edition por sólo 140 €, número de artículo 974; la Professional Edition cuesta 200 €, artículo 975. Tenemos el nuevo software de edición de escritorio de Adobe denominado InDesign por sólo 200 €, artículo 664. Ofrecemos los últimos juegos de Blizzard software. Puede empezar a jugar a Diablo II por sólo 30 €, número de artículo 12, y puede comprar Starcraft por sólo 10 €, número de artículo 812.”

1. Diséñese un documento HTML que muestre los artículos ofrecidos por Carahuevo.
2. Créese un documento XML bien formado que describa el contenido del catálogo de Carahuevo.
3. Créese un DTD para el documento XML y asegúrese de que el documento creado en la última pregunta es válido con respecto a ese DTD.
4. Escríbase una consulta XQuery que relacione todos los artículos de software del catálogo, ordenados por precio.
5. Escríbase una consulta XQuery que, para cada fabricante, relacione todos los artículos de software (es decir, una fila del resultado por fabricante).
6. Escríbase una consulta XQuery que relacione los precios de todos los artículos de hardware del catálogo.
7. Describanse los datos del catálogo en el modelo de datos semiestructurados como se ve en la Figura 18.7.
8. Créese una guía de datos para estos datos. Analícese la manera en que se puede utilizar (o no) para cada una de las consultas anteriores.
9. Diséñese un esquema relacional para editar estos datos.

Ejercicio 18.7 La base de datos de una universidad contiene información sobre los profesores y las asignaturas que imparten. La universidad ha decidido publicar esta información en Web y le han encargado su ejecución. Se le da la siguiente información sobre el contenido de la base de datos:

En el cuatrimestre de otoño de 1999 la asignatura “Introducción a los sistemas de gestión de bases de datos” la impartió el profesor Iniesto. La asignatura se impartía lunes y miércoles de 9 a 10 de la mañana en el aula 101. La parte de problemas se impartía los viernes de 9 a 10 de la mañana. En ese mismo cuatrimestre la asignatura “Sistemas avanzados de gestión de bases de datos” la impartió el profesor Carranza. Treinta y cinco alumnos se matricularon en esa asignatura, que se impartió en el aula 110 los martes y jueves de 1 a 2 de la tarde. En el cuatrimestre de primavera de 1999 la asignatura “Introducción a los sistemas de gestión de bases de datos” la impartió U.N. Ocón los martes y jueves de 3 a 4 de la tarde en el aula 110. Se matricularon sesenta y tres alumnos; la parte de problemas era los jueves de 4 a 5 de la tarde. La otra asignatura impartida en el cuatrimestre de primavera fue “Sistemas avanzados de gestión de bases de datos” por el profesor Iniesto, los lunes, miércoles y viernes de 8 a 9 de la mañana.

1. Créese un documento XML bien formado que contenga la base de datos de la universidad.

2. Créese un DTD para ese documento XML. Asegúrese de que el documento XML sea válido con respecto a ese DTD.
3. Escríbase una consulta de XQuery que relacione los nombres de todos los profesores en el orden en que aparecen en Web.
4. Escríbase una consulta XQuery que relacione todas las asignaturas impartidas en 1999. El resultado debe estar agrupado por profesor, con una fila por cada uno de ellos, y ordenado por apellidos. Para cada profesor los cursos deben estar ordenados por nombre y no deben contener valores duplicados (es decir, aunque algún profesor impartiera dos veces la misma asignatura en 1999, sólo debe aparecer una vez en el resultado).
5. Créese una guía de datos para estos datos. Analícese el modo en que se puede utilizar (o no) para cada una de las consultas anteriores.
6. Diséñese un esquema relacional para publicar estos datos.
7. Describese la información en un documento XML diferente —un documento que tenga una estructura diferente—. Créese el DTD correspondiente y asegúrese de que el documento sea válido. Vuelva a formular las consultas que escribió para partes anteriores de este ejercicio para que funcionen con el nuevo DTD.

Ejercicio 18.8 Considérese la base de datos del fabricante de ropa RopaFamiliar. RopaFamiliar produce tres tipos de ropa: ropa de mujer, ropa de hombre y ropa de niño. Los hombres pueden escoger entre polos y camisetas. Cada polo tiene una lista de colores y tallas disponibles y un precio uniforme. Cada camiseta tiene un precio, una lista de colores disponibles y una lista de tallas disponibles. Las mujeres tienen la misma posibilidad de elegir entre polos y camisetas que los hombres. Además, las mujeres pueden escoger entre tres tipos de vaqueros: talle esbelto, talle cómodo y talle relajado. Cada par de vaqueros tiene una lista de tallas de cintura posibles y de longitudes posibles. El precio de cada par de vaqueros depende sólo de su tipo. Los niños pueden escoger entre camisetas y gorras de béisbol. Cada camiseta tiene un precio, una lista de colores disponibles y una lista de dibujos disponibles. Todas las camisetas para niños tienen la misma talla. Las gorras de béisbol vienen en tres tallas diferentes: pequeña, mediana y grande. Cada artículo tiene un precio rebajado opcional que se oferta en oportunidades especiales. Escríbanse todas las consultas en XQuery.

1. Diséñese un DTD de XML para RopaFamiliar de modo que RopaFamiliar pueda publicar su catálogo en la Web.
2. Escríbase una consulta que determine el artículo más caro vendido por RopaFamiliar.
3. Escríbase una consulta que determine el precio promedio de cada tipo de ropa.
4. Escríbase una consulta que relacione todos los artículos que cuesten más que el promedio de su tipo; el resultado debe contener una fila por tipo en el orden en que se relacionan los tipos en Web. Para cada tipo, los artículos deben aparecer en orden de precios crecientes.
5. Escríbase una consulta que determine todos los artículos cuyo precio rebajado sea más del doble del precio normal de algún otro artículo.
6. Escríbase una consulta que determine todos los artículos cuyo precio rebajado sea más del doble del precio normal de algún otro artículo del mismo tipo de ropa.
7. Créese una guía de datos para esos datos. Analícese el modo en que se puede utilizar (o no) para cada una de las consultas anteriores.
8. Diséñese un esquema relacional para publicar estos datos.

Ejercicio 18.9 Supóngase que cada elemento e de un documento XML se asocia con la tripleta de números $\langle \text{comienzo}, \text{final}, \text{nivel} \rangle$, donde **comienzo** denota la posición inicial de e en el documento en términos de la salida de bits del archivo, **final** denota la posición final de ese elemento y **nivel** indica el nivel de anidamiento de e , donde el elemento raíz comienza en el nivel de anidamiento 0.

1. Exprésese la condición de que el elemento e_1 sea (i) un ancestro (ii) el padre del elemento e_2 en términos de esas tripletas.

2. Supóngase que cada elemento tiene un identificador interno generado por el sistema y que, para cada nombre de etiqueta q , se almacena la lista de identificadores de todos los elementos del documento que tienen la etiqueta q , es decir, una lista invertida de identificadores por etiqueta. Junto con el identificador del elemento también se almacena la tripleta asociada con él y se ordena la lista por la posición *comienzo* de los elementos. Ahora bien, supóngase que se desea evaluar la expresión de ruta $a//b$. El resultado de la reunión deben ser pares $\langle id_a, id_b \rangle$ tales que id_a e id_b sean identificadores de elementos e_a con el nombre de etiqueta a y e_b con el nombre de etiqueta b , respectivamente, y e_a sea ancestro de e_b . Debe estar ordenada según la clave compuesta \langle posición de *comienzo* de e_a , posición de *comienzo* de e_b \rangle . Diséñese un algoritmo que mezcle las listas de a y de b y lleve a cabo esa reunión. El número de comparaciones de posición debe ser lineal en el tamaño de la entrada y de la salida. *Sugerencia:* el enfoque es parecido a una ordenación-reunión de dos listas ordenadas de enteros.
3. Supóngase que se tienen k listas ordenadas de enteros, donde k es una constante. Supóngase que no hay valores duplicados; es decir, que cada valor aparece exactamente en una lista y exactamente una vez. Diséñese un algoritmo que mezcle estas listas en el que el número de comparaciones sea lineal en el tamaño de los datos.
4. A continuación supóngase que se desea llevar a cabo la reunión $a_1//a_2//\dots//a_k$ (una vez más, k es una constante). El resultado de la reunión debe ser una lista de k -tuplas $\langle id_1, id_2, \dots, id_k \rangle$ tal que id_i sea el identificador del elemento e_i con nombre de etiqueta a_i y e_i sea un ancestro de e_{i+1} para todo $1 \leq i \leq k-1$. La lista debe estar ordenada según la clave compuesta \langle posición de *comienzo* de e_1, \dots , posición de *comienzo* de e_k \rangle . Amplíense los algoritmos diseñados en las partes (2) y (3) para calcular esta reunión. El número de comparaciones de posición debe ser lineal en el tamaño combinado de los datos y de la salida.

Ejercicio 18.10 Este ejercicio examina el motivo de que la indexación de rutas para los datos XML sea diferente de los problemas convencionales de indexación, como la indexación de dominios ordenados linealmente para las consultas de puntos y distancias. Se ha propuesto el modelo siguiente para el problema de la indexación en general: los datos del problema consisten en (i) el dominio \mathcal{D} de elementos (ii) el ejemplar de datos I , que es un subconjunto finito de \mathcal{D} y (iii) un conjunto finito de consultas \mathcal{C} ; cada consulta es un subconjunto no vacío de I . Esta tripleta $\langle \mathcal{D}, I, \mathcal{C} \rangle$ representa la carga de trabajo indexada. El esquema de indexación \mathcal{E} para esta carga de trabajo agrupa fundamentalmente los elementos de datos en bloques de tamaño fijo B . Formalmente, \mathcal{E} es un conjunto de bloques $\{E_1, E_2, \dots, E_k\}$, donde cada bloque es un subconjunto de I que contiene exactamente B elementos. Estos bloques, conjuntamente, deben saturar I ; es decir, $I = E_1 \cup E_2 \dots \cup E_k$.

1. Supóngase que \mathcal{D} es el conjunto de enteros positivos y que I consiste en enteros desde 1 hasta n . \mathcal{C} consiste en todas las consultas de puntos; es decir, de los conjuntos unitarios $\{1\}, \{2\}, \dots, \{n\}$. Supóngase que se desea indexar esta carga de trabajo empleando un árbol B+ en el que cada bloque del nivel de hojas pueda albergar exactamente h enteros. ¿Cuál es el tamaño de bloque de este esquema de indexación? ¿Cuál es el número de bloques empleados?
2. La *redundancia de almacenamiento* del esquema de indexación \mathcal{E} es el número máximo de bloques que contienen un elemento de I . ¿Cuál es la redundancia de almacenamiento del árbol B+ empleado en la parte (1) anterior?
3. Defínase el *coste de acceso* de la consulta C de \mathcal{C} bajo el esquema \mathcal{E} para que sea el número mínimo de bloques de \mathcal{E} que la cubren. La sobrecarga de acceso de C es su coste de acceso dividido por su coste ideal de acceso, que es $\lceil |C|/B \rceil$. ¿Cuál es el coste de acceso de cualquier consulta bajo el esquema de árbol B+ de la parte (1)? ¿Qué ocurre con la sobrecarga de acceso?
4. La sobrecarga de acceso del propio esquema de indexación es la máxima sobrecarga de acceso entre todas las consultas de \mathcal{C} . Demuéstrese que este valor nunca puede ser superior a B . ¿Cuál es la sobrecarga de acceso del esquema de árbol B+?
5. A continuación se definirá una carga de trabajo para la indexación de rutas. El dominio $\mathcal{D} = \{i : i \text{ es un entero positivo}\}$. Intuitivamente se trata del conjunto de todos los identificadores de objetos. Las instancias pueden ser cualquier subconjunto finito de \mathcal{D} . Para definir \mathcal{C} , se impone una estructura de árbol al conjunto de identificadores de objetos de I . Por tanto, si hay n identificadores en I , se define el

árbol A con n nodos y se asocia cada nodo exactamente con un identificador de I . El árbol tiene raíz y se le etiquetan los nodos, para lo cual las etiquetas de los nodos proceden del conjunto infinito de etiquetas Σ . La raíz de A tiene la etiqueta distinguida denominada **raíz**. Ahora, C contiene un subconjunto S de los identificadores de objetos de I si S es el resultado de alguna expresión de ruta para A . La clase de expresiones de ruta que se está considerando sólo incluye a expresiones de ruta sencillas; es decir, expresiones de la forma $ER = \text{raízs}_1 t_1 s_2 t_2 \dots t_n$, donde cada s_i es un separador que puede ser bien $/$, bien $//$ y cada t_i es una etiqueta de Σ . Esta expresión devuelve el conjunto de todos los identificadores de objetos correspondientes a los nodos de A que tienen una ruta que coincide con ER y que se dirige a ellos.

Demuéstrese que, para cualquier r , hay una carga de trabajo de indexación de rutas tal que en cualquier esquema de indexación con redundancia, como máximo r tendrá una sobrecarga de acceso $B - 1$.

Ejercicio 18.11 Este ejercicio introduce el concepto de *simulación de grafos* en el contexto de la minimización de consultas. Considérese el siguiente tipo de restricciones para los datos: (1) Restricciones de padres obligatorias, en las que se puede especificar que el padre del elemento de etiqueta **b** tenga siempre la etiqueta **a** y (2) Restricciones de ancestros obligatorias, en las que se puede especificar que el elemento de etiqueta **b** tenga siempre un ancestro de etiqueta **a**.

1. Se representa la consulta de expresiones de ruta $ER = \text{raízs}_1 t_1 s_2 t_2 \dots t_n$, donde cada s_i es un separador y cada t_i es una etiqueta, como un grafo dirigido con un nodo para la **raíz** y otro para cada t_i . Los arcos van desde la **raíz** hasta t_1 , y desde t_i hasta t_{i+1} . Un arco es arco padre o ancestro según si su separador respectivo es $/$ o $//$. En el texto se representa el arco padre que va de u a v como $u \rightarrow v$ y el arco ancestro equivalente como $u \Rightarrow v$.

Como ejercicio sencillo, represéntese la expresión de ruta **raíz** $//a/b/c$ en forma de grafo.

2. Las restricciones se representan también como grafos dirigidos de la manera siguiente. Créese un nodo para cada nombre de etiqueta. Hay un parente (ancestro) desde el nombre de etiqueta **a** hasta el nombre de etiqueta **b** si existe una restricción que asegure que cada elemento **b** debe tener un parente (ancestro) **a**. Justifíquese por qué este grafo de restricciones debe ser acíclico para que las restricciones tengan sentido; es decir, para que haya ejemplares de datos que las satisfagan.
3. Las *simulaciones* son relaciones binarias \leq para los nodos de dos grafos acíclicos dirigidos y agrupados G_1 y G_2 que satisfacen la condición siguiente: Si $u \leq v$, donde u es un nodo de G_1 y v es un nodo de G_2 , para cada nodo $u' \rightarrow u$ debe existir $v' \rightarrow v$ tal que $u' \leq v'$ y, para cada $u'' \Rightarrow u$, debe existir v'' que sea ancestro de v (es decir, tenga alguna ruta hasta v) tal que $u'' \leq v''$. Demuéstrese que hay una única relación de simulación de mayor tamaño \leq^m . Si $u \leq^m v$, entonces se dice que u está *simulada por* v .
4. Demuéstrese que la expresión de ruta **raíz** $//b/c$ se puede reescribir como $//c$ si sólo si el nodo **c** del grafo de restricciones puede simular el nodo **c** del grafo de consultas.
5. La expresión de ruta $//t_j s_{j+1} t_{j+1} \dots t_n$ ($j > 1$) es un *sufijo* de la **raízs** $_1 t_1 s_2 t_2 \dots t_n$. Es un *sufijo equivalente* si sus resultados son iguales para todas las instancias de la base de datos que satisfacen las restricciones. Demuéstrese que esto ocurre si t_j del grafo de restricciones puede simular t_j del grafo de consultas.

NOTAS BIBLIOGRÁFICAS

Entre el material de lectura introductorio sobre la recuperación de información están los libros de texto estándar de Salton y McGill [385] y de van Rijsbergen [459]. Jones y Willett [263] y Frakes y Baeza-Yates [185] han editado conjuntos de artículos para los lectores más avanzados. La consulta de los repositorios de texto se ha estudiado ampliamente en el campo de la recuperación de la información; véase [376] para un resumen reciente. Faloutsos ofrece una visión general de los métodos de indexación para las bases de datos de texto [167]. Los archivos invertidos se estudian en [336] y los archivos de firmas en [168]. Zobel, Moffat y Ramamohanarao ofrecen una comparación de los archivos invertidos y de los archivos de firmas [492]. En [113] se presenta un resumen de las actualizaciones incrementales para los índices invertidos. Otros aspectos de la recuperación de información y de la indexación en el contexto de las bases de datos se abordan en [363],

[190], [392], y [493], entre otros. [215] estudia el problema del descubrimiento de recursos de texto en Web. El lirio de Witten, Moffat y Bell tiene mucho material sobre las técnicas de compresión para las bases de datos documentales [475].

El número de citas como medida del impacto científico lo estudió en primer lugar Garfield [200]; véase también [464]. El empleo de la información hipertextual para la mejora de la calidad de los motores de búsqueda lo han propuesto Spertus [422] y Weiss *et al.* [469]. El algoritmo HITS lo desarrolló Jon Kleinberg [279]. De manera concurrente, Brin y Page desarrollaron el algoritmo Pagerank (clasificación de páginas) (ahora denominado PigeonRank - clasificación de paloma), que también tiene en cuenta los hipervínculos entre páginas [68]. Puede verse un análisis completo y la comparación de varios algoritmos recientemente propuestos para la determinación de las páginas con autoridad en [62]. El descubrimiento de la estructura de World Wide Web constituye actualmente un área de investigación muy activa; véase, por ejemplo, el trabajo de Gibson *et al.* [209].

Hay mucha investigación sobre los datos semiestructurados en la comunidad de bases de datos. El sistema de integración de datos Tsimmis utiliza un modelo de datos semiestructurados para afrontar la posible heterogeneidad de los orígenes de datos [353, 352]. Se puede hallar trabajo sobre la descripción de la estructura de las bases de datos semiestructuradas en [341]. Wang y Liu consideran el descubrimiento de esquemas para los documentos semiestructurados [466]. La correspondencia entre las representaciones relacional y XML se estudia en [179, 405, 60] y [81].

Se han desarrollado varios lenguajes de consulta nuevos para los datos semiestructurados: LOREL [361], Quilt [95], UnQL [73], StruQL [178], WebSQL [328], y XML-QL [148]. La norma actual del W3C, XQuery, se describe en [96]. La última versión de varias normas mencionadas en este capítulo - incluidos XML, XSchema, XPath y XQuery - se puede hallar en la página Web del Consorcio World Wide Web (World Wide Web Consortium, W3C) (www.w3.org). Kweelt [384] es un sistema de código abierto que soporta Quilt, y es una plataforma adecuada para la experimentación de sistemas que se puede obtener en línea de <http://kweelt.sourceforge.net>.

LORE es un sistema de gestión de bases de datos diseñado para los datos semiestructurados [323]. La optimización de consultas para los datos semiestructurados se aborda en [3] y en [212], que propuso la guía fuerte de datos. El índice 1 se propuso en [335] para abordar el problema de la explosión de tamaños para las guías de datos. Otro esquema de indexación XML se propone en [128]. Los trabajos más recientes [268] tienden a ampliar el entramado de los índices de estructuras para que abarque subconjuntos concretos de expresiones de ruta. La estimación de selectividad de las expresiones de ruta XML se examina en [4]. La teoría de la indexabilidad propuesta por Hellerstein *et al.* en [248] permite un análisis formal del problema de la indexación de rutas, que resulta ser más difícil que la indexación tradicional.

Ha habido mucho trabajo sobre el empleo de los modelos de datos semiestructurados para los datos en Web y se han desarrollado varios sistemas de consulta de la Web: WebSQL [328], W3QS [283], WebLog [289], WebOQL [24], STRUDEL [177], ARANEUS [30] y FLORID [251]. [183] es una buena visión general de la investigación sobre bases de datos en el contexto Web.

BIBLIOGRAFÍA

- [1] S. Abiteboul, R. Hull y V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [2] S. Abiteboul y P. Kanellakis. Object identity as a query language primitive. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1989.
- [3] S. Abiteboul y V. Vianu. Regular path queries with constraints. En *Proc. ACM Symp. on Principles of Database Systems*, 1997.
- [4] A. Aboulnaga, A. R. Alameldeen y J. F. Naughton. Estimating the selectivity of XML path expressions for Internet scale applications. En *Proceedings of VLDB*, 2001.
- [5] S. Acharya, P. B. Gibbons, V. Poosala y S. Ramaswamy. The Aqua approximate query answering system. En *Proc. ACM SIGMOD Conf. on the Management of Data*, páginas 574–576. ACM Press, 1999.
- [6] S. Acharya, P. B. Gibbons, V. Poosala y S. Ramaswamy. Join synopses for approximate query answering. En *Proc. ACM SIGMOD Conf. on the Management of Data*, páginas 275–286. ACM Press, 1999.
- [7] S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J. Naughton, R. Ramakrishnan y S. Sarawagi. On the computation of multidimensional aggregates. En *Proc. Intl. Conf. on Very Large Databases*, 1996.
- [8] R. C. Agarwal, C. C. Aggarwal y V. V. V. Prasad. A tree projection algorithm for generation of frequent item sets. *Journal of Parallel and Distributed Computing*, 61(3):350–371, 2001.
- [9] R. Agrawal y N. Gehani. ODE (Object Database and Environment): The language and the data model. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1989.
- [10] R. Agrawal, J. E. Gehrke, D. Gunopulos y P. Raghavan. Automatic subspace clustering of high dimensional data for data mining. En *Proc. ACM SIGMOD Conf. on Management of Data*, 1998.
- [11] R. Agrawal, T. Imielinski y A. Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, December 1993.
- [12] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen y A. I. Verkamo. Fast discovery of association rules. En U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth y R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, capítulo 12, páginas 307–328. AAAI/MIT Press, 1996.
- [13] R. Agrawal, G. Psaila, E. Wimmers y M. Zaot. Querying shapes of histories. En *Proc. Intl. Conf. on Very Large Databases*, 1995.
- [14] R. Agrawal y J. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962–969, 1996.
- [15] R. Agrawal y R. Srikant. Mining sequential patterns. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1995.
- [16] R. Agrawal, P. Stolorz y G. Piatetsky-Shapiro, editors. *Proc. Intl. Conf. on Knowledge Discovery and Data Mining*. AAAI Press, 1998.
- [17] A. Aho, C. Beeri y J. Ullman. The theory of joins in relational databases. *ACM Transactions on Database Systems*, 4(3):297–314, 1979.
- [18] A. Aho, J. Hopcroft y J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1983.
- [19] A. Aiken, J. Widom y J. Hellerstein. Static analysis techniques for predicting the behavior of active database rules. *ACM Transactions on Database Systems*, 20(1):3–41, 1995.

- [20] N. Alon, P. B. Gibbons, Y. Matias y M. Szegedy. Tracking join and self-join sizes in limited storage. En *Proc. ACM Symposium on Principles of Database Systems*, Philadelphia, Pennsylvania, 1999.
- [21] N. Alon, Y. Matias y M. Szegedy. The space complexity of approximating the frequency moments. En *Proc. of the ACM Symp. on Theory of Computing*, páginas 20–29, 1996.
- [22] E. Anwar, L. Maugis y U. Chakravarthy. A new perspective on rule support for object-oriented databases. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1993.
- [23] W. Armstrong. Dependency structures of database relationships. En *Proc. IFIP Congress*, 1974.
- [24] G. Arocena y A. O. Mendelzon. WebOQL: restructuring documents, databases and webs. En *Proc. Intl. Conf. on Data Engineering*, 1988.
- [25] M. Astrahan, M. Blasgen, D. Chamberlin, K. Eswaran, J. Gray, P. Griffiths, W. King, R. Lorie, P. McJones, J. Mehl, G. Putzolu, I. Traiger, B. Wade y V. Watson. System R: a relational approach to database management. *ACM Transactions on Database Systems*, 1(2):97–137, 1976.
- [26] M. Atkinson, P. Bailey, K. Chisholm, P. Cockshott y R. Morrison. An approach to persistent programming. En *Readings in Object-Oriented Databases*. eds. S.B. Zdonik and D. Maier, Morgan Kaufmann, 1990.
- [27] M. Atkinson y P. Buneman. Types and persistence in database programming languages. *ACM Computing Surveys*, 19(2):105–190, 1987.
- [28] P. Atzeni, L. Cabibbo y G. Mecca. Isalog: A declarative language for complex objects with hierarchies. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1993.
- [29] P. Atzeni y V. De Antonellis. *Relational Database Theory*. Benjamin-Cummings, 1993.
- [30] P. Atzeni, G. Mecca y P. Merialdo. To weave the web. En *Proc. Intl. Conf. Very Large Data Bases*, 1997.
- [31] R. Avnur, J. Hellerstein, B. Lo, C. Olston, B. Raman, V. Raman, T. Roth y K. Wylie. Control: Continuous output and navigation technology with refinement online. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1998.
- [32] R. Avnur y J. M. Hellerstein. Eddies: Continuously adaptive query processing. En *Proc. ACM SIGMOD Conf. on the Management of Data*, páginas 261–272. ACM, 2000.
- [33] B. Babcock, S. Babu, M. Datar, R. Motwani y J. Widom. Models and issues in data stream systems. En *Proc. ACM Symp. on Principles of Database Systems*, 2002.
- [34] S. Babu y J. Widom. Continous queries over data streams. *ACM SIGMOD Record*, 30(3):109–120, 2001.
- [35] D. Badal y G. Popek. Cost and performance analysis of semantic integrity validation methods. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1979.
- [36] A. Badia, D. Van Gucht y M. Gyssens. Querying with generalized quantifiers. En *Applications of Logic Databases*. ed. R. Ramakrishnan, Kluwer Academic, 1995.
- [37] F. Bancilhon, C. Delobel y P. Kanellakis. *Building an Object-Oriented Database System*. Morgan Kaufmann, 1991.
- [38] F. Bancilhon y S. Khoshafian. A calculus for complex objects. *Journal of Computer and System Sciences*, 38(2):326–340, 1989.
- [39] F. Bancilhon y N. Spryatos. Update semantics of relational views. *ACM Transactions on Database Systems*, 6(4):557–575, 1981.
- [40] E. Baralis, S. Ceri y S. Paraboschi. Modularization techniques for active rules design. *ACM Transactions on Database Systems*, 21(1):1–29, 1996.
- [41] D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. T. Ng, V. Poosala, K. A. Ross y K. C. Sevcik. The New Jersey data reduction report. *Data Engineering Bulletin*, 20(4):3–45, 1997.

- [42] R. Barquin y H. Edelstein. *Planning and Designing the Data Warehouse*. Prentice-Hall, 1997.
- [43] C. Batini, S. Ceri y S. Navathe. *Database Design: An Entity Relationship Approach*. Benjamin/Cummings Publishers, 1992.
- [44] C. Batini, M. Lenzerini y S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
- [45] R. J. Bayardo. Efficiently mining long patterns from databases. En *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, páginas 85–93. ACM Press, 1998.
- [46] R. J. Bayardo, R. Agrawal y D. Gunopulos. Constraint-based rule mining in large, dense databases. *Data Mining and Knowledge Discovery*, 4(2/3):217–240, 2000.
- [47] R. Bayer y E. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189, 1972.
- [48] C. Beeri, R. Fagin y J. Howard. A complete axiomatization of functional and multivalued dependencies in database relations. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1977.
- [49] C. Beeri y P. Honeyman. Preserving functional dependencies. *SIAM Journal of Computing*, 10(3):647–656, 1982.
- [50] C. Beeri y T. Milo. A model for active object-oriented database. En *Proc. Intl. Conf. on Very Large Databases*, 1991.
- [51] J. Bentley y J. Friedman. Data structures for range searching. *ACM Computing Surveys*, 13(3):397–409, 1979.
- [52] P. Bernstein. Synthesizing third normal form relations from functional dependencies. *ACM Transactions on Database Systems*, 1(4):277–298, 1976.
- [53] P. Bernstein, B. Blaustein y E. Clarke. Fast maintenance of semantic integrity assertions using redundant aggregate data. En *Proc. Intl. Conf. on Very Large Databases*, 1980.
- [54] P. Bernstein y E. Newcomer. *Principles of Transaction Processing*. Morgan Kaufmann, 1997.
- [55] P. Bernstein, D. Shipman y W. Wong. Formal aspects of serializability in database concurrency control. *IEEE Transactions on Software Engineering*, 5(3):203–216, 1979.
- [56] K. Beyer y R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1999.
- [57] J. Biskup y B. Convent. A formal view integration method. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1986.
- [58] J. Biskup, U. Dayal y P. Bernstein. Synthesizing independent database schemas. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1979.
- [59] J. Blakeley, P.-A. Larson y F. Tompa. Efficiently updating materialized views. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1986.
- [60] P. Bohannon, J. Freire, P. Roy y J. Simeon. From XML schema to relations: A cost-based approach to XML storage. En *Proceedings of ICDE*, 2002.
- [61] P. Bonnet y D. E. Shasha. *Database Tuning: Principles, Experiments, and Troubleshooting Techniques*. Morgan Kaufmann Publishers, 2002.
- [62] A. Borodin, G. Roberts, J. Rosenthal y P. Tsaparas. Finding authorities and hubs from link structures on Roberts G.O. the world wide web. En *World Wide Web Conference*, páginas 415–429, 2001.
- [63] R. Boyce y D. Chamberlin. SEQUEL: A structured English query language. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1974.
- [64] P. S. Bradley y U. M. Fayyad. Refining initial points for K-Means clustering. En *Proc. Intl. Conf. on Machine Learning*, páginas 91–99. Morgan Kaufmann, San Francisco, CA, 1998.

- [65] P. S. Bradley, U. M. Fayyad y C. Reina. Scaling clustering algorithms to large databases. En *Proc. Intl. Conf. on Knowledge Discovery and Data Mining*, 1998.
- [66] L. Breiman, J. H. Friedman, R. A. Olshen y C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont. CA, 1984.
- [67] S. Brin, R. Motwani y C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1997.
- [68] S. Brin y L. Page. The anatomy of a large-scale hypertextual web search engine. En *Proceedings of 7th World Wide Web Conference*, 1998.
- [69] S. Brin, R. Motwani, J. D. Ullman y S. Tsur. Dynamic itemset counting and implication rules for market basket data. En *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, páginas 255–264. ACM Press, 1997.
- [70] N. Bruno, S. Chaudhuri y L. Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Transactions on Database Systems*, To appear, 2002.
- [71] F. Bry y R. Manthey. Checking consistency of database constraints: A logical basis. En *Proc. Intl. Conf. on Very Large Databases*, 1986.
- [72] P. Buneman y E. Clemons. Efficiently monitoring relational databases. *ACM Transactions on Database Systems*, 4(3), 1979.
- [73] P. Buneman, S. Davidson, G. Hillebrand y D. Suciu. A query language and optimization techniques for unstructured data. En *Proc. ACM SIGMOD Conf. on Management of Data*, 1996.
- [74] P. Buneman, S. Naqvi, V. Tannen y L. Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149(1):3–48, 1995.
- [75] D. Burdick, M. Calimlim y J. E. Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. En *Proc. Intl. Conf. on Data Engineering (ICDE)*. IEEE Computer Society, 2001.
- [76] M. Carey, D. Chamberlin, S. Narayanan, B. Vance, D. Doole, S. Rielau, R. Swagerman y N. Mattos. O-O, what's happening to DB2? En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1999.
- [77] M. Carey, D. DeWitt, M. Franklin, N. Hall, M. McAuliffe, J. Naughton, D. Schuh, M. Solomon, C. Tan, O. Tsatalos, S. White y M. Zwilling. Shoring up persistent applications. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1994.
- [78] M. Carey, D. DeWitt, G. Graefe, D. Haught, J. Richardson, D. Schuh, E. Shekita y S. Vandenberg. The EXODUS Extensible DBMS project: An overview. En S. Zdonik y D. Maier, editors, *Readings in Object-Oriented Databases*. Morgan Kaufmann, 1990.
- [79] M. Carey, D. DeWitt y J. Naughton. The 007 benchmark. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1993.
- [80] M. Carey, D. DeWitt, J. Naughton, M. Asgarian, J. Gehrke y D. Shah. The BUCKY object-relational benchmark. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1997.
- [81] M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Shanmugasundaram, E. Shekita y S. Subramanian. XPERANTO: publishing object-relational data as XML. En *Proceedings of the Third International Workshop on the Web and Databases*, May 2000.
- [82] M. Carey y D. Kossman. On saying “Enough Already!” in SQL. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1997.
- [83] M. Carey y D. Kossman. Reducing the braking distance of an SQL query engine. En *Proc. Intl. Conf. on Very Large Databases*, 1998.
- [84] M. Casanova, L. Tucherman y A. Furtado. Enforcing inclusion dependencies and referential integrity. En *Proc. Intl. Conf. on Very Large Databases*, 1988.
- [85] M. Casanova y M. Vidal. Towards a sound view integration methodology. En *ACM Symp. on Principles of Database Systems*, 1983.

- [86] S. Castano, M. Fugini, G. Martella y P. Samarati. *Database Security*. Addison-Wesley, 1995.
- [87] R. Cattell. *The Object Database Standard: ODMG-93 (Release 1.1)*. Morgan Kaufmann, 1994.
- [88] S. Ceri, P. Fraternali, S. Paraboschi y L. Tanca. Active rule management in Chimera. En J. Widom y S. Ceri, editors, *Active Database Systems*. Morgan Kaufmann, 1996.
- [89] S. Ceri y J. Widom. Deriving production rules for constraint maintenance. En *Proc. Intl. Conf. on Very Large Databases*, 1990.
- [90] F. Cesarini, M. Missikoff y G. Soda. An expert system approach for database application tuning. *Data and Knowledge Engineering*, 8:35–55, 1992.
- [91] U. Chakravarthy. Architectures and monitoring techniques for active databases: An evaluation. *Data and Knowledge Engineering*, 16(1):1–26, 1995.
- [92] D. Chamberlin. *Using the New DB2*. Morgan Kaufmann, 1996.
- [93] D. Chamberlin, M. Astrahan, M. Blasgen, J. Gray, W. King, B. Lindsay, R. Lorie, J. Mehl, T. Price, P. Selinger, M. Schkolnick, D. Slutz, I. Traiger, B. Wade y R. Yost. A history and evaluation of System R. *Communications of the ACM*, 24(10):632–646, 1981.
- [94] D. Chamberlin, M. Astrahan, K. Eswaran, P. Griffiths, R. Lorie, J. Mehl, P. Reisner y B. Wade. Sequel 2: a unified approach to data definition, manipulation, and control. *IBM Journal of Research and Development*, 20(6):560–575, 1976.
- [95] D. Chamberlin, D. Florescu y J. Robie. Quilt: an XML query language for heterogeneous data sources. En *Proceedings of WebDB*, Dallas, TX, May 2000.
- [96] D. Chamberlin, D. Florescu, J. Robie, J. Simeon y M. Stefanescu. XQuery: A query language for XML. World Wide Web Consortium, <http://www.w3.org/TR/xquery>, Feb 2000.
- [97] D. Chang y D. Harkey. *Client/ server data access with Java and XML*. John Wiley and Sons, 1998.
- [98] M. Charikar, S. Chaudhuri, R. Motwani y V. R. Narasayya. Towards estimation error guarantees for distinct values. En *Proc. ACM Symposium on Principles of Database Systems*, páginas 268–279. ACM, 2000.
- [99] D. Chatziantoniou y K. Ross. Groupwise processing of relational queries. En *Proc. Intl. Conf. on Very Large Databases*, 1997.
- [100] S. Chaudhuri y U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74, 1997.
- [101] S. Chaudhuri y D. Madigan, editors. *Proc. ACM SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*. ACM Press, 1999.
- [102] S. Chaudhuri y V. Narasayya. An efficient cost-driven index selection tool for Microsoft SQL Server. En *Proc. Intl. Conf. on Very Large Databases*, 1997.
- [103] S. Chaudhuri y V. R. Narasayya. Autoadmin 'what-if' index analysis utility. En *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 1998.
- [104] S. Chaudhuri y K. Shim. Optimization of queries with user-defined predicates. En *Proc. Intl. Conf. on Very Large Databases*, 1996.
- [105] S. Chaudhuri y K. Shim. Optimization queries with aggregate views. En *Intl. Conf. on Extending Database Technology*, 1996.
- [106] S. Chaudhuri, G. Das y V. R. Narasayya. A robust, optimization-based approach for approximate answering of aggregate queries. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 2001.
- [107] P. P. Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.
- [108] Y. Chen, G. Dong, J. Han, B. W. Wah y J. Wang. Multi-dimensional regression analysis of time-series data streams. En *Proc. Intl. Conf. on Very Large Data Bases*, 2002.

630 Sistemas de gestión de bases de datos

- [109] D. W. Cheung, J. Han, V. T. Ng y C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. En *Proc. Int. Conf. Data Engineering*, 1996.
- [110] D. W. Cheung, V. T. Ng y B. W. Tam. Maintenance of discovered knowledge: A case in multi-level association rules. En *Proc. Intl. Conf. on Knowledge Discovery and Data Mining*. AAAI Press, 1996.
- [111] D. Childs. Feasibility of a set theoretical data structure —A general structure based on a reconstructed definition of relation. *Proc. Tri-annual IFIP Conference*, 1968.
- [112] F. Chin y G. Ozsoyoglu. Statistical database design. *ACM Transactions on Database Systems*, 6(1):113–139, 1981.
- [113] T.-C. Chiueh y L. Huang. Efficient real-time index updates in text retrieval systems.
- [114] J. Chomicki. Real-time integrity constraints. En *ACM Symp. on Principles of Database Systems*, 1992.
- [115] P. Chrysanthis y K. Ramamritham. Acta: A framework for specifying and reasoning about transaction structure and behavior. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1990.
- [116] F. Civelek, A. Dogac y S. Spaccapietra. An expert system approach to view definition and integration. En *Proc. Entity-Relationship Conference*, 1988.
- [117] R. Cochrane, H. Pirahesh y N. Mattos. Integrating triggers and declarative constraints in SQL database systems. En *Proc. Intl. Conf. on Very Large Databases*, 1996.
- [118] CODASYL. *Report of the CODASYL Data Base Task Group*. ACM, 1971.
- [119] E. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.
- [120] E. Codd. Further normalization of the data base relational model. En R. Rustin, editor, *Data Base Systems*. Prentice Hall, 1972.
- [121] E. Codd. Relational completeness of data base sub-languages. En R. Rustin, editor, *Data Base Systems*. Prentice Hall, 1972.
- [122] E. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.
- [123] E. Codd. Twelve rules for on-line analytic processing. *Computerworld*, April 13 1995.
- [124] L. Colby, T. Griffin, L. Libkin, I. Mumick y H. Trickey. Algorithms for deferred view maintenance. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1996.
- [125] L. Colby, A. Kawaguchi, D. Lieuwen, I. Mumick y K. Ross. Supporting multiple view maintenance policies: Concepts, algorithms, and performance analysis. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1997.
- [126] D. Comer. The ubiquitous B-tree. *ACM C. Surveys*, 11(2):121–137, 1979.
- [127] D. Connolly, editor. *XML Principles, Tools and Techniques*. O'Reilly & Associates, Sebastopol, USA, 1997.
- [128] B. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason y M. Shadmon. A fast index for semistructured data. En *Proceedings of VLDB*, 2001.
- [129] D. Copeland y D. Maier. Making SMALLTALK a database system. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1984.
- [130] G. Cornell y K. Abdali. *CGI Programming With Java*. Prentice-Hall, 1998.
- [131] C. Cortes, K. Fisher, D. Pregibon y A. Rogers. Hancock: a language for extracting signatures from data streams. En *Proc. ACM SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*, páginas 9–17. AAAI Press, 2000.
- [132] J. Daemen y V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard (Information Security and Cryptography)*. Springer Verlag, 2002.

- [133] M. Datar, A. Gionis, P. Indyk y R. Motwani. Maintaining stream statistics over sliding windows. En *Proc. of the Annual ACM-SIAM Symp. on Discrete Algorithms*, 2002.
- [134] C. Date. A critique of the SQL database language. *ACM SIGMOD Record*, 14(3):8–54, 1984.
- [135] C. Date. *Relational Database: Selected Writings*. Addison-Wesley, 1986.
- [136] C. Date. *An Introduction to Database Systems*. Addison-Wesley, 6.^a edición, 1995.
- [137] C. Date. *An Introduction to Database Systems*. Addison-Wesley, 7.^a edición, 1999.
- [138] C. Date y R. Fagin. Simple conditions for guaranteeing higher normal forms in relational databases. *ACM Transactions on Database Systems*, 17(3), 1992.
- [139] C. Date y D. McGoveran. *A Guide to Sybase and SQL Server*. Addison-Wesley, 1993.
- [140] U. Dayal y P. Bernstein. On the correct translation of update operations on relational views. *ACM Transactions on Database Systems*, 7(3), 1982.
- [141] P. DeBra y J. Paredaens. Horizontal decompositions for handling exceptions to FDs. En H. Gallaire, J. Minker y J.-M. Nicolas, editors, *Advances in Database Theory*. Plenum Press, 1981.
- [142] J. Deep y P. Holfelder. *Developing CGI applications with Perl*. Wiley, 1996.
- [143] C. Delobel. Normalization and hierachial dependencies in the relational data model. *ACM Transactions on Database Systems*, 3(3):201–222, 1978.
- [144] D. Denning. Secure statistical databases with random sample queries. *ACM Transactions on Database Systems*, 5(3):291–315, 1980.
- [145] D. E. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.
- [146] A. Deshpande. An implementation for nested relational databases. Informe técnico, PhD thesis, Indiana University, 1989.
- [147] P. Deshpande, K. Ramasamy, A. Shukla y J. F. Naughton. Caching multidimensional queries using chunks. En *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 1998.
- [148] A. Deutsch, M. Fernandez, D. Florescu, A. Levy y D. Suciu. XML-QL: A query language for XML. World Wide Web Consortium, <http://www.w3.org/TR/NOTE-xml-ql>, Aug 1998.
- [149] O. e. a. Deux. The story of O2. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), 1990.
- [150] O. Diaz, N. Paton y P. Gray. Rule management in object-oriented databases: A uniform approach. En *Proc. Intl. Conf. on Very Large Databases*, 1991.
- [151] W. Diffie y M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
- [152] P. Domingos y G. Hulten. Mining high-speed data streams. En *Proc. ACM SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*. AAAI Press, 2000.
- [153] D. Donjerkovic y R. Ramakrishnan. Probabilistic optimization of top N queries. En *Proc. Intl. Conf. on Very Large Databases*, 1999.
- [154] R. C. Dubes y A. Jain. *Clustering Methodologies in Exploratory Data Analysis, Advances in Computers*. Academic Press, New York, 1980.
- [155] A. Eisenberg y J. Melton. SQL:1999, formerly known as SQL 3 *ACM SIGMOD Record*, 28(1):131–138, 1999.
- [156] R. Elmasri y S. Navathe. Object integration in database design. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1984.
- [157] R. Elmasri y S. Navathe. *Fundamentals of Database Systems*. Benjamin-Cummings, 3.^a edición, 2000.
- [158] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer y X. Xu. Incremental clustering for mining in a data warehousing environment. En *Proc. Intl. Conf. On Very Large Data Bases*, 1998.

- [159] M. Ester, H.-P. Kriegel, J. Sander y X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. En *Proc. Intl. Conf. on Knowledge Discovery in Databases and Data Mining*, 1995.
- [160] M. Ester, H.-P. Kriegel y X. Xu. A database interface for clustering in large spatial databases. En *Proc. Intl. Conf. on Knowledge Discovery in Databases and Data Mining*, 1995.
- [161] K. Eswaran y D. Chamberlin. Functional specification of a subsystem for data base integrity. En *Proc. Intl. Conf. on Very Large Databases*, 1975.
- [162] K. Eswaran, J. Gray, R. Lorie y I. Traiger. The notions of consistency and predicate locks in a data base system. *Communications of the ACM*, 19(11):624–633, 1976.
- [163] R. Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems*, 2(3):262–278, 1977.
- [164] R. Fagin. Normal forms and relational database operators. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1979.
- [165] R. Fagin. A normal form for relational databases that is based on domains and keys. *ACM Transactions on Database Systems*, 6(3):387–415, 1981.
- [166] R. Fagin, J. Nievergelt, N. Pippenger y H. Strong. Extendible Hashing —a fast access method for dynamic files. *ACM Transactions on Database Systems*, 4(3), 1979.
- [167] C. Faloutsos. Access methods for text. *ACM Computing Surveys*, 17(1):49–74, 1985.
- [168] C. Faloutsos y S. Christodoulakis. Signature files: An access method for documents and its analytical performance evaluation. *ACM Transactions on Office Information Systems*, 2(4):267–288, 1984.
- [169] C. Faloutsos y H. Jagadish. On B-Tree indices for skewed distributions. En *Proc. Intl. Conf. on Very Large Databases*, 1992.
- [170] C. Faloutsos, M. Ranganathan y Y. Manolopoulos. Fast subsequence matching in time-series databases. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1994.
- [171] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani y J. D. Ullman. Computing iceberg queries efficiently. En *Proc. Intl. Conf. On Very Large Data Bases*, 1998.
- [172] U. Fayyad, G. Piatetsky-Shapiro y P. Smyth. The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34, 1996.
- [173] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth y R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. MIT Press, 1996.
- [174] U. Fayyad y E. Simoudis. Data mining and knowledge discovery: Tutorial notes. En *Intl. Joint Conf. on Artificial Intelligence*, 1997.
- [175] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth y R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.
- [176] U. M. Fayyad y R. Uthurusamy, editors. *Proc. Intl. Conf. on Knowledge Discovery and Data Mining*. AAAI Press, 1995.
- [177] M. Fernandez, D. Florescu, J. Kang, A. Y. Levy y D. Suciu. STRUDEL: A Web site management system. En *Proc. ACM SIGMOD Conf. on Management of Data*, 1997.
- [178] M. Fernandez, D. Florescu, A. Y. Levy y D. Suciu. A query language for a Web -site management system. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 26(3):4–11, 1997.
- [179] M. Fernandez, D. Suciu y W. Tan. SilkRoute: trading between relations and XML. En *Proceedings of the WWW9*, 2000.
- [180] S. Finkelstein, M. Schkolnick y P. Tiberio. Physical database design for relational databases. *IBM Research Review RJ5034*, 1986.

- [181] D. Fishman, D. Beech, H. Cate, E. Chow, T. Connors, J. Davis, N. Derrett, C. Hoch, W. Kent, P. Lyngbaek, B. Mahbod, M.-A. Neimat, T. Ryan y M.-C. Shan. Iris: an object-oriented database management system *ACM Transactions on Office Information Systems*, 5(1):48–69, 1987.
- [182] C. Fleming y B. von Halle. *Handbook of Relational Database Design*. Addison-Wesley, 1989.
- [183] D. Florescu, A. Y. Levy y A. O. Mendelzon. Database techniques for the World-Wide Web: A survey. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 27(3):59–74, 1998.
- [184] W. Ford y M. S. Baum. *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption (2nd Edition)*. Prentice Hall, 2000.
- [185] W. B. Frakes y R. Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.
- [186] M. Franklin, M. Carey y M. Livny. Local disk caching for client-server database systems. En *Proc. Intl. Conf. on Very Large Databases*, 1993.
- [187] P. Fraternali y L. Tanca. A structured approach for the definition of the semantics of active databases. *ACM Transactions on Database Systems*, 20(4):414–471, 1995.
- [188] O. Friesen, A. Lefebvre y L. Vieille. VALIDITY: Applications of a DOOD system. En *Intl. Conf. on Extending Database Technology*, 1996.
- [189] J. Fry y E. Sibley. Evolution of data-base management systems. *ACM Computing Surveys*, 8(1):7–42, 1976.
- [190] N. Fuhr. A decision-theoretic approach to database selection in networked ir. *ACM Transactions on Database Systems*, 17(3), 1999.
- [191] T. Fukuda, Y. Morimoto, S. Morishita y T. Tokuyama. Mining optimized association rules for numeric attributes. En *ACM Symp. on Principles of Database Systems*, 1996.
- [192] A. Furtado y M. Casanova. Updating relational views. En *Query Processing in Database Systems*. eds. W. Kim, D.S. Reiner and D.S. Batory, Springer-Verlag, 1985.
- [193] H. Gallaire, J. Minker y J.-M. Nicolas (eds.). *Advances in Database Theory, Vols. 1 and 2*. Plenum Press, 1984.
- [194] H. Gallaire y J. Minker (eds.). *Logic and Data Bases*. Plenum Press, 1978.
- [195] V. Ganti, J. Gehrke y R. Ramakrishnan. Demon: mining and monitoring evolving data. *IEEE Transactions on Knowledge and Data Engineering*, 13(1), 2001.
- [196] V. Ganti, J. Gehrke, R. Ramakrishnan y W.-Y. Loh. Focus: a framework for measuring changes in data characteristics. En *Proc. ACM Symposium on Principles of Database Systems*, 1999.
- [197] V. Ganti, J. E. Gehrke y R. Ramakrishnan. Cactus-clustering categorical data using summaries. En *Proc. ACM Intl. Conf. on Knowledge Discovery in Databases*, 1999.
- [198] V. Ganti, R. Ramakrishnan, J. E. Gehrke, A. Powell y J. French. Clustering large datasets in arbitrary metric spaces. En *Proc. IEEE Intl. Conf. Data Engineering*, 1999.
- [199] H. Garcia-Molina, J. Ullman y J. Widom. *Database Systems: The Complete Book* Prentice Hall, 2001.
- [200] E. Garfield. Citation analysis as a tool in journal evaluation. *Science*, 178(4060):471–479, 1972.
- [201] A. Garg y C. Gotlieb. Order preserving key transformations. *ACM Transactions on Database Systems*, 11(2):213–234, 1986.
- [202] J. E. Gehrke, V. Ganti, R. Ramakrishnan y W.-Y. Loh. Boat: Optimistic decision tree construction. En *Proc. ACM SIGMOD Conf. on Management of Data*, 1999.
- [203] J. E. Gehrke, F. Korn y D. Srivastava. On computing correlated aggregates over continual data streams. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 2001.

- [204] J. E. Gehrke, R. Ramakrishnan y V. Ganti. Rainforest: A framework for fast decision tree construction of large datasets. En *Proc. Intl. Conf. on Very Large Databases*, 1998.
- [205] S. P. Ghosh. *Data Base Organization for Data Management (2nd ed.)*. Academic Press, 1986.
- [206] P. B. Gibbons, Y. Matias y V. Poosala. Fast incremental maintenance of approximate histograms. En *Proc. of the Conf. on Very Large Databases*, 1997.
- [207] P. B. Gibbons y Y. Matias. New sampling-based summary statistics for improving approximate query answers. En *Proc. ACM SIGMOD Conf. on the Management of Data*, páginas 331–342. ACM Press, 1998.
- [208] D. Gibson, J. M. Kleinberg y P. Raghavan. Clustering categorical data: An approach based on dynamical systems. En *Proc. Intl. Conf. Very Large Data Bases*, 1998.
- [209] D. Gibson, J. M. Kleinberg y P. Raghavan. Inferring web communities from link topology. En *Proc. ACM Conf. on Hypertext*, 1998.
- [210] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan y M. J. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. En *Proc. of the Conf. on Very Large Databases*, 2001.
- [211] C. F. Goldfarb y P. Prescod. *The XML Handbook*. Prentice-Hall, 1998.
- [212] R. Goldman y J. Widom. DataGuides: enabling query formulation and optimization in semistructured databases. En *Proc. Intl. Conf. on Very Large Data Bases*, páginas 436–445, 1997.
- [213] M. Graham, A. Mendelzon y M. Vardi. Notions of dependency satisfaction. *Journal of the ACM*, 33(1): 105–129, 1986.
- [214] G. Grahne. *The Problem of Incomplete Information in Relational Databases*. Springer-Verlag, 1991.
- [215] L. Gravano, H. Garcia-Molina y A. Tomasic. Gloss: text-source discovery over the internet. *ACM Transactions on Database Systems*, 24(2), 1999.
- [216] J. Gray. The transaction concept: Virtues and limitations. En *Proc. Intl. Conf. on Very Large Databases*, 1981.
- [217] J. Gray. *The Benchmark Handbook: for Database and Transaction Processing Systems*. Morgan Kaufmann, 1991.
- [218] J. Gray, A. Bosworth, A. Layman y H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1996.
- [219] J. Gray y A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1992.
- [220] P. Gray. *Logic, Algebra, and Databases*. John Wiley, 1984.
- [221] M. Greenwald y S. Khanna. Space-efficient online computation of quantile summaries. En *Proc. ACM SIGMOD Conf. on Management of Data*, 2001.
- [222] P. Griffiths y B. Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems*, 1(3):242–255, 1976.
- [223] S. Guha, N. Mishra, R. Motwani y L. O'Callaghan. Clustering data streams. En *Proc. of the Annual Symp. on Foundations of Computer Science*, 2000.
- [224] S. Guha, R. Rastogi y K. Shim. Cure: an efficient clustering algorithm for large databases. En *Proc. ACM SIGMOD Conf. on Management of Data*, 1998.
- [225] S. Guha, N. Koudas y K. Shim. Data streams and histograms. En *Proc. of the ACM Symp. on Theory of Computing*, 2001.
- [226] D. Gunopulos, H. Mannila, R. Kharden y H. Toivonen. Data mining, hypergraph transversals, and machine learning. En *Proc. ACM Symposium on Principles of Database Systems*, páginas 209–216, 1997.
- [227] D. Gunopulos, H. Mannila y S. Saluja. Discovering all most specific sentences by randomized algorithms. En *Proc. of the Intl. Conf. on Database Theory*, volumen 1186 de *Lecture Notes in Computer Science*, páginas 215–229. Springer, 1997.

- [228] A. Gupta y I. Mumick. *Materialized Views: Techniques, Implementations, and Applications* MIT Press, 1999.
- [229] A. Gupta, I. Mumick y V. Subrahmanian. Maintaining views incrementally. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1993.
- [230] L. Haas, W. Chang, G. Lohman, J. McPherson, P. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M. Carey y E. Shekita. Starburst mid-flight: As the dust clears. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), 1990.
- [231] P. J. Haas y J. M. Hellerstein. Ripple joins for online aggregation. En *Proc. ACM SIGMOD Conf. on the Management of Data*, páginas 287–298. ACM Press, 1999.
- [232] M. Hall. *Core Web Programming: HTML , Java , CGI , & Javascript*. Prentice-Hall, 1997.
- [233] G. Hamilton, R. G. Cattell y M. Fisher. *JDBC Database Access With Java: A Tutorial and Annotated Reference*. Java Series. Addison-Wesley, 1997.
- [234] M. Hammer y D. McLeod. Semantic integrity in a relational data base system. En *Proc. Intl. Conf. on Very Large Databases*, 1975.
- [235] J. Han y Y. Fu. Discovery of multiple-level association rules from large databases. En *Proc. Intl. Conf. on Very Large Databases*, 1995.
- [236] D. Hand. *Construction and Assessment of Classification Rules*. John Wiley & Sons, Chichester, England, 1997.
- [237] J. Han y M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.
- [238] J. Han, J. Pei y Y. Yin. Mining frequent patterns without candidate generation. En *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, páginas 1–12, 2000.
- [239] E. Hanson. A performance analysis of view materialization strategies. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1987.
- [240] E. Hanson. Rule condition testing and action execution in Ariel. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1992.
- [241] V. Harinarayan, A. Rajaraman y J. Ullman. Implementing data cubes efficiently. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1996.
- [242] J. Harrison y S. Dietrich. Maintenance of materialized views in deductive databases: An update propagation approach. En *Proc. Workshop on Deductive Databases*, 1992.
- [243] T. Hastie, R. Tibshirani y J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Verlag, 2001.
- [244] D. Heckerman. Bayesian networks for knowledge discovery. En *Advances in Knowledge Discovery and Data Mining*. eds. U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, MIT Press, 1996.
- [245] D. Heckerman, H. Mannila, D. Pregibon y R. Uthurusamy, editors. *Proc. Intl. Conf. on Knowledge Discovery and Data Mining*. AAAI Press, 1997.
- [246] J. Hellerstein. Optimization and execution techniques for queries with expensive methods. *Ph.D. thesis, University of Wisconsin-Madison*, 1995.
- [247] J. Hellerstein, P. Haas y H. Wang. Online aggregation. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1997.
- [248] J. Hellerstein, E. Koutsoupias y C. Papadimitriou. On the analysis of indexing schemes. En *Proceedings of the ACM Symposium on Principles of Database Systems*, páginas 249–256. ACM Press, 1997.
- [249] J. Hellerstein, J. Naughton y A. Pfeffer. Generalized search trees for database systems. En *Proc. Intl. Conf. on Very Large Databases*, 1995.
- [250] C. Hidber. Online association rule mining. En *Proc. ACM SIGMOD Conf. on the Management of Data*, páginas 145–156, 1999.

- [251] R. Himmeroeder, G. Lausen, B. Ludaescher y C. Schlepphorst. On a declarative semantics for Web queries. *Lecture Notes in Computer Science*, 1341:386–398, 1997.
- [252] C.-T. Ho, R. Agrawal, N. Megiddo y R. Srikant. Range queries in OLAP data cubes. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1997.
- [253] S. Holzner. *XML Complete*. McGraw-Hill, 1998.
- [254] R. Hull y R. King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19(19):201–260, 1987.
- [255] R. Hull y J. Su. Algebraic and calculus query languages for recursively typed complex objects. *Journal of Computer and System Sciences*, 47(1):121–156, 1993.
- [256] R. Hull y M. Yoshikawa. ILOG: Declarative creation and manipulation of object-identifiers. En *Proc. Intl. Conf. on Very Large Databases*, 1990.
- [257] G. Hulten, L. Spencer y P. Domingos. Mining time-changing data streams. En *Proc. ACM SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*, páginas 97–106. AAAI Press, 2001.
- [258] J. Hunter. *Java Servlet Programming*. O'Reilly Associates, Inc., 1998.
- [259] T. Imielinski y W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.
- [260] T. Imielinski y H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 38(11):58–64, 1996.
- [261] A. K. Jain y R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.
- [262] S. Jajodia y R. Sandhu. Polyinstantiation integrity in multilevel relations. En *Proc. IEEE Symp. on Security and Privacy*, 1990.
- [263] K. S. Jones y P. Willett, editors. *Readings in Information Retrieval*. Multimedia Information and Systems. Morgan Kaufmann Publishers, 1997.
- [264] J. Jou y P. Fischer. The complexity of recognizing 3NF schemes. *Information Processing Letters*, 14(4):187–190, 1983.
- [265] P. Kanellakis. Elements of relational database theory. En *Handbook of Theoretical Computer Science*. ed. J. Van Leeuwen, Elsevier, 1991.
- [266] H. Kargupta y P. Chan, editors. *Advances in Distributed and Parallel Knowledge Discovery*. MIT Press, 2000.
- [267] L. Kaufman y P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley and Sons, 1990.
- [268] R. Kaushik, P. Bohannon, J. F. Naughton y H. F. Korth. Covering indexes for branching path expression queries. En *Proceedings of SIGMOD*, 2002.
- [269] A. Keller. Algorithms for translating view updates to database updates for views involving selections, projections, and joins. *ACM Symp. on Principles of Database Systems*, 1985.
- [270] W. Kent. *Data and Reality, Basic Assumptions in Data Processing Reconsidered*. North-Holland, 1978.
- [271] L. Kerschberg, A. Klug y D. Tsichritzis. A taxonomy of data models. En *Systems for Large Data Bases*. eds. P.C. Lockemann and E.J. Neuhold, North-Holland, 1977.
- [272] M. Kifer, W. Kim y Y. Sagiv. Querying object-oriented databases. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1992.
- [273] M. Kifer, G. Lausen y J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.
- [274] W. Kim. Object-oriented database systems: Promise, reality, and future. En *Proc. Intl. Conf. on Very Large Databases*, 1993.

- [275] W. Kim, J. Garza, N. Ballou y D. Woelk. Architecture of the ORION next-generation database system. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):109–124, 1990.
- [276] W. Kim y F. Lochovsky (eds.). *Object-Oriented Concepts, Databases, and Applications*. Addison-Wesley, 1989.
- [277] W. Kim (ed.). *Modern Database Systems*. ACM Press and Addison-Wesley, 1995.
- [278] R. Kimball. *The Data Warehouse Toolkit*. John Wiley and Sons, 1996.
- [279] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. En *Proc. ACM -SIAM Symp. on Discrete Algorithms*, 1998.
- [280] A. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM*, 29(3):699–717, 1982.
- [281] D. Knuth. *The Art of Computer Programming, Vol. 3 —Sorting and Searching*. Addison-Wesley, 1973.
- [282] G. Koch y K. Loney. *Oracle: The Complete Reference*. Oracle Press, Osborne-McGraw-Hill, 1995.
- [283] D. Konopnicki y O. Shmueli. W3QS: A system for WWW querying. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1997.
- [284] F. Korn, H. Jagadish y C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. En *Proc. ACM SIGMOD Conf. on Management of Data*, 1997.
- [285] M. Kornacker, C. Mohan y J. Hellerstein. Concurrency and recovery in generalized search trees. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1997.
- [286] Y. Kotidis y N. Roussopoulos. An alternative storage organization for ROLAP aggregate views based on cubetrees. En *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 1998.
- [287] J. Kuhns. Logical aspects of question answering by computer. Informe técnico, Rand Corporation, RM-5428-Pr., 1967.
- [288] M. LaCroix y A. Pirotte. Domain oriented relational languages. En *Proc. Intl. Conf. on Very Large Databases*, 1977.
- [289] L. Lakshmanan, F. Sadri y I. N. Subramanian. A declarative query language for querying and restructuring the web. En *Proc. Intl. Conf. on Research Issues in Data Engineering*, 1996.
- [290] L. V. S. Lakshmanan, Raymond T. Ng, J. Han y A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. En *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, páginas 157–168. ACM Press, 1999.
- [291] C. Lam, G. Landis, J. Orenstein y D. Weinreb. The Objectstore database system. *Communications of the ACM*, 34(10), 1991.
- [292] C. Landwehr. Formal models of computer security. *ACM Computing Surveys*, 13(3):247–278, 1981.
- [293] R. Langerak. View updates in relational databases with an independent scheme. *ACM Transactions on Database Systems*, 15(1):40–66, 1990.
- [294] P.-A. Larson. Linear hashing with overflow-handling by linear probing. *ACM Transactions on Database Systems*, 10(1):75–89, 1985.
- [295] P.-A. Larson. Linear hashing with separators — A dynamic hashing scheme achieving one-access retrieval. *ACM Transactions on Database Systems*, 13(3):366–388, 1988.
- [296] T. Leung y R. Muntz. Temporal query processing and optimization in multiprocessor database machines. En *Proc. Intl. Conf. on Very Large Databases*, 1992.
- [297] M. Leventhal, D. Lewis y M. Fuchs. *Designing XML Internet applications*. The Charles F. Goldfarb series on open information management. Prentice-Hall, 1998.
- [298] P. Lewis, A. Bernstein y M. Kifer. *Databases and Transaction Processing*. Addison Wesley, 2001.
- [299] D.-I. Lin y Z. M. Kedem. Pincer search: A new algorithm for discovering the maximum frequent set. *Lecture Notes in Computer Science*, 1377:105–??, 1998.

- [300] V. Linnemann, K. Kuspert, P. Dadam, P. Pistor, R. Erbe, A. Kemper, N. Sudkamp, G. Walch y M. Wallrath. Design and implementation of an extensible database management system supporting user defined data types and functions. En *Proc. Intl. Conf. on Very Large Databases*, 1988.
- [301] B. Liskov, A. Adya, M. Castro, M. Day, S. Ghemawat, R. Gruber, U. Maheshwari, A. Myers y L. Shriram. Safe and efficient sharing of persistent objects in Thor. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1996.
- [302] W. Litwin. Linear Hashing: A new tool for file and table addressing. En *Proc. Intl. Conf. on Very Large Databases*, 1980.
- [303] W. Litwin. Trie Hashing. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1981.
- [304] W. Litwin, M.-A. Neimat y D. Schneider. LH * — A scalable, distributed data structure. *ACM Transactions on Database Systems*, 21(4):480–525, 1996.
- [305] Y. Lou y Z. Ozsoyoglu. LLO: An object-oriented deductive language with methods and method inheritance. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1991.
- [306] C. Lucchesi y S. Osborn. Candidate keys for relations. *J. Computer and System Sciences*, 17(2):270–279, 1978.
- [307] V. Lum. Multi-attribute retrieval with combined indexes. *Communications of the ACM*, 1(11):660–665, 1970.
- [308] T. Lunt, D. Denning, R. Schell, M. Heckman y W. Shockley. The seaview security model. *IEEE Transactions on Software Engineering*, 16(6):593–607, 1990.
- [309] D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.
- [310] D. Maier, A. Mendelzon y Y. Sagiv. Testing implication of data dependencies. *ACM Transactions on Database Systems*, 4(4), 1979.
- [311] A. Makinouchi. A consideration on normal form of not-necessarily-normalized relation in the relational data model. En *Proc. Intl. Conf. on Very Large Databases*, 1977.
- [312] G. Manku, S. Rajagopalan y B. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. En *Proc. ACM SIGMOD Conf. on Management of Data*, 1999.
- [313] H. Mannila. Methods and problems in data mining. En *Intl. Conf. on Database Theory*, 1997.
- [314] H. Mannila y K.-J. Raiha. Design by Example: An application of Armstrong relations. *Journal of Computer and System Sciences*, 33(2):126–141, 1986.
- [315] H. Mannila y K.-J. Raiha. *The Design of Relational Databases*. Addison-Wesley, 1992.
- [316] H. Mannila, H. Toivonen y A. I. Verkamo. Discovering frequent episodes in sequences. En *Proc. Intl. Conf. on Knowledge Discovery in Databases and Data Mining*, 1995.
- [317] H. Mannila, P. Smyth y D. J. Hand. *Principles of Data Mining*. MIT Press, 2001.
- [318] V. Markowitz. Representing processes in the extended entity-relationship model. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1990.
- [319] V. Markowitz. Safe referential integrity structures in relational databases. En *Proc. Intl. Conf. on Very Large Databases*, 1991.
- [320] Y. Matias, J. S. Vitter y M. Wang. Dynamic maintenance of wavelet-based histograms. En *Proc. of the Conf. on Very Large Databases*, 2000.
- [321] D. McCarthy y U. Dayal. The architecture of an active data base management system. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1989.
- [322] W. McCune y L. Henschen. Maintaining state constraints in relational databases: A proof theoretic basis. *Journal of the ACM*, 36(1):46–68, 1989.

- [323] J. McHugh, S. Abiteboul, R. Goldman, D. Quass y J. Widom. Lore: A database management system for semistructured data. *ACM SIGMOD Record*, 26(3):54–66, 1997.
- [324] M. Mehta, R. Agrawal y J. Rissanen. SLIQ: A fast scalable classifier for data mining. En *Proc. Intl. Conf. on Extending Database Technology*, 1996.
- [325] J. Melton. *Advanced SQL:1999, Understanding Object-Relational and Other Advanced Features*. Morgan Kaufmann, 2002.
- [326] J. Melton y A. Simon. *Understanding the New SQL: A Complete Guide*. Morgan Kaufmann, 1993.
- [327] J. Melton y A. Simon. *SQL:1999, Understanding Relational Language Components*. Morgan Kaufmann, 2002.
- [328] A. O. Mendelzon, G. A. Mihaila y T. Milo. Querying the World Wide Web. *Journal on Digital Libraries*, 1:54–67, 1997.
- [329] R. Meo, G. Psaila y S. Ceri. A new SQL -like operator for mining association rules. En *Proc. Intl. Conf. on Very Large Databases*, 1996.
- [330] T. Merrett. The extended relational algebra, a basis for query languages. En *Databases*. ed. Shneiderman, Academic Press, 1978.
- [331] T. Merrett. *Relational Information Systems*. Reston Publishing Company, 1983.
- [332] D. Michie, D. Spiegelhalter y C. Taylor, editors. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, London, 1994.
- [333] Microsoft. *Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference*. Microsoft Press, 1997.
- [334] R. Miller, Y. Ioannidis y R. Ramakrishnan. The use of information capacity in schema integration and translation. En *Proc. Intl. Conf. on Very Large Databases*, 1993.
- [335] T. Milo y D. Suciu. Index structures for path expressions. En *ICDT: 7th International Conference on Database Theory*, 1999.
- [336] A. Moffat y J. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379, 1996.
- [337] A. Motro. Superviews: Virtual integration of multiple databases. *IEEE Transactions on Software Engineering*, 13(7):785–798, 1987.
- [338] A. Motro y P. Buneman. Constructing superviews. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1981.
- [339] I. Mumick y K. Ross. Noodle: A language for declarative querying in an object-oriented database. En *Intl. Conf. on Deductive and Object-Oriented Databases*, 1993.
- [340] M. Negri, G. Pelagatti y L. Sbattella. Formal semantics of SQL queries. *ACM Transactions on Database Systems*, 16(3), 1991.
- [341] S. Nestorov, J. Ullman, J. Weiner y S. Chawathe. Representative objects: Concise representations of semi-structured, hierarchical data. En *Proc. Intl. Conf. on Data Engineering*. IEEE Computer Society, 1997.
- [342] R. T. Ng y J. Han. Efficient and effective clustering methods for spatial data mining. En *Proc. Intl. Conf. on Very Large Databases*, Santiago, Chile, September 1994.
- [343] R. T. Ng, L. V. S. Lakshmanan, J. Han y A. Pang. Exploratory mining and pruning optimizations of constrained association rules. En *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, páginas 13–24. ACM Press, 1998.
- [344] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha y R. Motwani. Streaming-data algorithms for high-quality clustering. En *Proc. of the Intl. Conference on Data Engineering*. IEEE, 2002.
- [345] F. Olken y D. Rotem. Maintenance of materialized views of sampling queries. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1992.

- [346] P. O'Neil. *Database Principles, Programming, and Practice*. Morgan Kaufmann, 1994.
- [347] P. O'Neil y E. O'Neil. *Database Principles, Programming, and Performance*. Addison Wesley, 2.^a edición, 2000.
- [348] P. O'Neil y D. Quass. Improved query performance with variant indexes. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1997.
- [349] G. Ozsoyoglu, Z. Ozsoyoglu y V. Matos. Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. *ACM Transactions on Database Systems*, 12(4):566–592, 1987.
- [350] Z. Ozsoyoglu y L.-Y. Yuan. A new normal form for nested relations. *ACM Transactions on Database Systems*, 12(1):111–136, 1987.
- [351] C. Papadimitriou. The serializability of concurrent database updates. *Journal of the ACM*, 26(4):631–653, 1979.
- [352] Y. Papakonstantinou, S. Abiteboul y H. Garcia-Molina. Object fusion in mediator systems. En *Proc. Intl. Conf. on Very Large Data Bases*, 1996.
- [353] Y. Papakonstantinou, H. Garcia-Molina y J. Widom. Object exchange across heterogeneous information sources. En *Proc. Intl. Conf. on Data Engineering*, 1995.
- [354] J. Peckham y F. Maryanski. Semantic data models. *ACM Computing Surveys*, 20(3):153–189, 1988.
- [355] J. Pei y J. Han. Can we push more constraints into frequent pattern mining? En *ACM SIGKDD Conference*, páginas 350–354, 2000.
- [356] J. Pei, J. Han y L. V. S. Lakshmanan. Mining frequent item sets with convertible constraints. En *Proc. Intl. Conf. on Data Engineering (ICDE)*, páginas 433–442. IEEE Computer Society, 2001.
- [357] S. Petrov. Finite axiomatization of languages for representation of system properties. *Information Sciences*, 47:339–372, 1989.
- [358] G. Piatetsky-Shapiro y W. J. Frawley, editors. *Knowledge Discovery in Databases*. AAAI/MIT Press, Menlo Park, CA, 1991.
- [359] N. Pitts-Moultis y C. Kirk. *XML black book: Indispensable problem solver*. Coriolis Group, 1998.
- [360] X.-L. Qian y G. Wiederhold. Incremental recomputation of active relational expressions. *IEEE Transactions on Knowledge and Data Engineering*, 3(3):337–341, 1990.
- [361] D. Quass, A. Rajaraman, Y. Sagiv y J. Ullman. Querying semistructured heterogeneous information. En *Proc. Intl. Conf. on Deductive and Object-Oriented Databases*, 1995.
- [362] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1993.
- [363] H. G. M. R. Alonso, D. Barbara. Data caching issues in an information retrieval system. *ACM Transactions on Database Systems*, 15(3), 1990.
- [364] D. Rafiei y A. Mendelzon. Similarity-based queries for time series data. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1997.
- [365] M. Ramakrishna. An exact probability model for finite hash tables. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1988.
- [366] M. Ramakrishna y P.-A. Larson. File organization using composite perfect hashing. *ACM Transactions on Database Systems*, 14(2):231–263, 1989.
- [367] R. Ramakrishnan, D. Donjerkovic, A. Ranganathan, K. Beyer y M. Krishnaprasad. SRQL: Sorted relational query language. En *Proc. IEEE Intl. Conf. on Scientific and Statistical DBMS*, 1998.
- [368] R. Ramakrishnan, S. Stolfo, R. J. Bayardo y I. Parsa, editors. *Proc. ACM SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*. AAAI Press, 2000.
- [369] K. Ramamohanarao, J. Shepherd y R. Sacks-Davis. Partial-match retrieval for dynamic files using linear hashing with partial expansions. En *Intl. Conf. on Foundations of Data Organization and Algorithms*, 1989.

- [370] V. Raman, B. Raman y J. M. Hellerstein. Online dynamic reordering for interactive data processing. En *Proc. of the Conf. on Very Large Databases*, páginas 709–720. Morgan Kaufmann, 1999.
- [371] S. Rao, A. Badia y D. Van Gucht. Providing better support for a class of decision support queries. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1996.
- [372] R. Rastogi y K. Shim. Public: A decision tree classifier that integrates building and pruning. En *Proc. Intl. Conf. on Very Large Databases*, 1998.
- [373] G. Reese. *Database Programming With JDBC and Java*. O'Reilly & Associates, 1997.
- [374] R. Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *Journal of the ACM*, 33(2):349–370, 1986.
- [375] E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison Wesley Professional, 2000.
- [376] E. Riloff y L. Hollaar. Text databases and information retrieval. En *Handbook of Computer Science*. ed. A.B. Tucker, CRC Press, 1996.
- [377] J. Rissanen. Independent components of relations. *ACM Transactions on Database Systems*, 2(4):317–325, 1977.
- [378] R. Rivest. Partial match retrieval algorithms. *SIAM Journal on Computing*, 5(1):19–50, 1976.
- [379] R. L. Rivest, A. Shamir y L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [380] K. Ross y D. Srivastava. Fast computation of sparse datacubes. En *Proc. Intl. Conf. on Very Large Databases*, 1997.
- [381] K. Ross, D. Srivastava y S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1996.
- [382] N. Roussopoulos, Y. Kotidis y M. Roussopoulos. Cubetree: Organization of and bulk updates on the data cube. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1997.
- [383] S. Rozen y D. Shasha. Using feature set compromise to automate physical database design. En *Proc. Intl. Conf. on Very Large Databases*, 1991.
- [384] A. Sahuguet, L. Dupont y T. Nguyen. Kweelt: Querying XML in the new millenium. <http://kweelt.sourceforge.net>, Sept 2000.
- [385] G. Salton y M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.
- [386] B. J. Salzberg. *File Structures*. Prentice-Hall, 1988.
- [387] J. Sander, M. Ester, H.-P. Kriegel y X. Xu. Density-based clustering in spatial databases. *J. of Data Mining and Knowledge Discovery*, 2(2), 1998.
- [388] R. E. Sanders. *ODBC 3.5 Developer's Guide*. McGraw-Hill Series on Data Warehousing and Data Management. McGraw-Hill, 1998.
- [389] S. Sarawagi y M. Stonebraker. Efficient organization of large multidimensional arrays. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1994.
- [390] S. Sarawagi, S. Thomas y R. Agrawal. Integrating mining with relational database systems: Alternatives and implications. En *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 1998.
- [391] A. Savasere, E. Omiecinski y S. Navathe. An efficient algorithm for mining association rules in large databases. En *Proc. Intl. Conf. on Very Large Databases*, 1995.
- [392] P. Schuble. Spider: A multiuser information retrieval system for semistructured and dynamic data. En *Proc. ACM SIGIR Conference on Research and Development in Information Retrieval*, páginas 318–327, 1993.
- [393] H.-J. Schek, H.-B. Paul, M. Scholl y G. Weikum. The DASDBS project: Objects, experiences, and future projects. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), 1990.
- [394] M. Schkolnick. Physical database design techniques. En *NYU Symp. on Database Design*, 1978.

- [395] M. Schkolnick y P. Sorenson. The effects of denormalization on database performance. Informe técnico, IBM RJ3082, San Jose, CA, 1981.
- [396] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, 1995.
- [397] E. Sciore. A complete axiomatization of full join dependencies. *Journal of the ACM*, 29(2):373–393, 1982.
- [398] A. Segev y J. Park. Maintaining materialized views in distributed databases. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1989.
- [399] A. Segev y A. Shoshani. Logical modeling of temporal data. *Proc. ACM SIGMOD Conf. on the Management of Data*, 1987.
- [400] P. Selfridge, D. Srivastava y L. Wilson. IDEA: Interactive data exploration and analysis. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1996.
- [401] P. Seshadri, M. Livny y R. Ramakrishnan. The design and implementation of a sequence database system. En *Proc. Intl. Conf. on Very Large Databases*, 1996.
- [402] P. Seshadri, M. Livny y R. Ramakrishnan. The case for enhanced abstract data types. En *Proc. Intl. Conf. on Very Large Databases*, 1997.
- [403] J. Shafer y R. Agrawal. SPRINT: a scalable parallel classifier for data mining. En *Proc. Intl. Conf. on Very Large Databases*, 1996.
- [404] J. Shanmugasundaram, U. Fayyad y P. Bradley. Compressed data cubes for olap aggregate query approximation on continuous dimensions. En *Proc. Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, 1999.
- [405] J. Shanmugasundaram, J. Kiernan, E. J. Shekita, C. Fan y J. Funderburk. Querying XML views of relational data. En *Proc. Intl. Conf. on Very Large Data Bases*, 2001.
- [406] D. Shasha. *Database Tuning: A Principled Approach*. Prentice-Hall, 1992.
- [407] D. Shasha, E. Simon y P. Valduriez. Simple rational guidance for chopping up transactions. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1992.
- [408] H. Shatkay y S. Zdonik. Approximate queries and representations for large data sequences. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1996.
- [409] T. Sheard y D. Stemple. Automatic verification of database transaction safety. *ACM Transactions on Database Systems*, 1989.
- [410] P. Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa y D. Shah. Turbo-charging vertical mining of large databases. En *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, páginas 22–33, May 2000.
- [411] A. Sheth, J. Larson, A. Cornelio y S. Navathe. A tool for integrating conceptual schemas and user views. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1988.
- [412] A. Shoshani. OLAP and statistical databases: Similarities and differences. En *ACM Symp. on Principles of Database Systems*, 1997.
- [413] A. Shukla, P. Deshpande, J. Naughton y K. Ramasamy. Storage estimation for multidimensional aggregates in the presence of hierarchies. En *Proc. Intl. Conf. on Very Large Databases*, 1996.
- [414] A. Silberschatz, H. Korth y S. Sudarshan. *Database System Concepts (4th ed.)*. McGraw-Hill, 4.^a edición, 2001.
- [415] E. Simon, J. Kiernan y C. de Maindreville. Implementing high-level active rules on top of relational databases. En *Proc. Intl. Conf. on Very Large Databases*, 1992.
- [416] E. Simoudis, J. Wei y U. M. Fayyad, editors. *Proc. Intl. Conf. on Knowledge Discovery and Data Mining*. AAAI Press, 1996.

- [417] J. Smith y D. Smith. Database abstractions: Aggregation and generalization. *ACM Transactions on Database Systems*, 1(1):105–133, 1977.
- [418] K. Smith y M. Winslett. Entity modeling in the MLS relational model. En *Proc. Intl. Conf. on Very Large Databases*, 1992.
- [419] P. Smith y M. Barnes. *Files and Databases: An Introduction*. Addison-Wesley, 1987.
- [420] S. Spaccapietra, C. Parent y Y. Dupont. Model independent assertions for integration of heterogeneous schemas. En *Proc. Intl. Conf. on Very Large Databases*, 1992.
- [421] S. Spaccapietra (ed.). *Entity-Relationship Approach: Ten Years of Experience in Information Modeling*, *Proc. Entity-Relationship Conf.* North-Holland, 1987.
- [422] E. Spertus. ParaSite: mining structural information on the web. En *Intl. World Wide Web Conference*, 1997.
- [423] R. Srikant y R. Agrawal. Mining generalized association rules. En *Proc. Intl. Conf. on Very Large Databases*, 1995.
- [424] R. Srikant y R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. En *Proc. ACM SIGMOD Conf. on Management of Data*, 1996.
- [425] R. Srikant y R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. En *Proc. Intl. Conf. on Extending Database Technology*, 1996.
- [426] R. Srikant, Q. Vu y R. Agrawal. Mining Association Rules with Item Constraints. En *Proc. Intl. Conf. on Knowledge Discovery in Databases and Data Mining*, 1997.
- [427] D. Srivastava, S. Dar, H. Jagadish y A. Levy. Answering queries with aggregation using views. En *Proc. Intl. Conf. on Very Large Databases*, 1996.
- [428] D. Srivastava, R. Ramakrishnan, P. Seshadri y S. Sudarshan. Coral++: Adding object-orientation to a logic database language. En *Proc. Intl. Conf. on Very Large Databases*, 1993.
- [429] J. Srivastava y D. Rotem. Analytical modeling of materialized view maintenance. En *ACM Symp. on Principles of Database Systems*, 1988.
- [430] J. Srivastava, J. Tan y V. Lum. Tbsam: An access method for efficient processing of statistical queries. *IEEE Transactions on Knowledge and Data Engineering*, 1(4):414–423, 1989.
- [431] P. Stachour y B. Thuraisingham. Design of LDV: A multilevel secure relational database management system. *IEEE Transactions on Knowledge and Data Engineering*, 2(2), 1990.
- [432] T. Steel. Interim report of the ANSI-SPARC study group. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1975.
- [433] M. Stonebraker. Implementation of integrity constraints and views by query modification. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1975.
- [434] M. Stonebraker. Inclusion of new types in relational database systems. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1986.
- [435] M. Stonebraker. *The INGRES Papers: Anatomy of a Relational Database System*. Addison-Wesley, 1986.
- [436] M. Stonebraker. *Object-relational DBMSs — The Next Great Wave*. Morgan Kaufmann, 1996.
- [437] M. Stonebraker, J. Frew, K. Gardels y J. Meredith. The Sequoia 2000 storage benchmark. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1993.
- [438] M. Stonebraker y J. Hellerstein (eds.). *Readings in Database Systems*. Morgan Kaufmann, 2.^a edición, 1994.
- [439] M. Stonebraker, A. Jhingran, J. Goh y S. Potamianos. On rules, procedures, caching and views in data base systems. En *UCBERL M9036*, 1990.

- [440] M. Stonebraker y G. Kemnitz. The Postgres next-generation database management system. *Communications of the ACM*, 34(10):78–92, 1991.
- [441] B. Subramanian, T. Leung, S. Vandenberg y S. Zdonik. The AQUA approach to querying lists and trees in object-oriented databases. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1995.
- [442] T. Teorey. *Database Modeling and Design: The E-R Approach*. Morgan Kaufmann, 1990.
- [443] T. Teorey, D.-Q. Yang y J. Fry. A logical database design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18(2):197–222, 1986.
- [444] S. A. Thomas. *SSL & TLS Essentials: Securing the Web*. John Wiley & Sons, 2000.
- [445] S. Thomas, S. Bodagala, K. Alsabti y S. Ranka. An efficient algorithm for the incremental updation of association rules in large databases. En *Proc. Intl. Conf. on Knowledge Discovery and Data Mining*. AAAI Press, 1997.
- [446] S. Todd. The Peterlee relational test vehicle. *IBM Systems Journal*, 15(4):285–307, 1976.
- [447] H. Toivonen. Sampling large databases for association rules. En *Proc. Intl. Conf. on Very Large Databases*, 1996.
- [448] TP Performance Council. TPC Benchmark D: Standard specification, rev. 1.2. Informe técnico, <http://www.tpc.org/dspec.html>, 1996.
- [449] M. Tsangaris y J. Naughton. On the performance of object clustering techniques. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1992.
- [450] D.-M. Tsou y P. Fischer. Decomposition of a relation scheme into Boyce-C odd normal form. *SIGACT News*, 14(3):23–29, 1982.
- [451] D. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov y A. Rosenthal. Query flocks: A generalization of association-rule mining. En *Proc. ACM SIGMOD Conf. on Management of Data*, páginas 1–12, 1998.
- [452] A. Tucker (ed.). *Computer Science and Engineering Handbook*. CRC Press, 1996.
- [453] J. W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.
- [454] J. Ullman. The U.R. strikes back. En *ACM Symp. on Principles of Database Systems*, 1982.
- [455] J. Ullman. *Principles of Database and Knowledgebase Systems, Vols. 1 and 2*. Computer Science Press, 1989.
- [456] J. Ullman. Information integration using logical views. En *Intl. Conf. on Database Theory*, 1997.
- [457] S. Urban y L. Delcambre. An analysis of the structural, dynamic, and temporal aspects of semantic data models. En *Proc. IEEE Intl. Conf. on Data Engineering*, 1986.
- [458] G. Valentin, M. Zuliani, D. C. Zilio, G. M. Lohman y A. Skelley. Db2 advisor: An optimizer smart enough to recommend its own indexes. En *Proc. Intl. Conf. on Data Engineering (ICDE)*, páginas 101–110. IEEE Computer Society, 2000.
- [459] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, London, United Kingdom, 1990.
- [460] M. Vardi. Incomplete information and default reasoning. En *ACM Symp. on Principles of Database Systems*, 1986.
- [461] M. Vardi. Fundamentals of dependency theory. En *Trends in Theoretical Computer Science*. ed. E. Borger, Computer Science Press, 1987.
- [462] J. S. Vitter y M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. En *Proc. ACM SIGMOD Conf. on the Management of Data*, páginas 193–204. ACM Press, 1999.
- [463] G. Vossen. *Data Models, Database Languages and Database Management Systems*. Addison-Wesley, 1991.
- [464] N. Wade. Citation analysis: A new tool for science administrators. *Science*, 188(4183):429–432, 1975.

- [465] R. Wagner. Indexing design considerations. *IBM Systems Journal*, 12(4):351–367, 1973.
- [466] K. Wang y H. Liu. Schema discovery for semistructured data. En *Third International Conference on Knowledge Discovery and Data Mining (KDD -97)*, páginas 271–274, 1997.
- [467] G. Weddell. Reasoning about functional dependencies generalized for semantic data models. *ACM Transactions on Database Systems*, 17(1), 1992.
- [468] G. Weikum y G. Vossen. *Transactional Information Systems*. Morgan Kaufmann, 2001.
- [469] R. Weiss, B. V. lez, M. A. Sheldon, C. Manprempre, P. Szilagyi, A. Duda y D. K. Gifford. HyPursuit: A hierarchical network search engine that exploits content-link hypertext clustering. En *Proc. ACM Conf. on Hypertext*, 1996.
- [470] S. White, M. Fisher, R. Cattell, G. Hamilton y M. Hapner. *JDBC API Tutorial and Reference: Universal Data Access for the Java 2 Platform*. Addison-Wesley, 2.^a edición, 1999.
- [471] J. Widom y S. Ceri. *Active Database Systems*. Morgan Kaufmann, 1996.
- [472] G. Wiederhold. *Database Design (2nd ed.)*. McGraw-Hill, 1983.
- [473] M. S. Winslett. A model-based approach to updating databases with incomplete information. *ACM Transactions on Database Systems*, 13(2):167–196, 1988.
- [474] G. Wiorkowski y D. Kull. *DB2: Design and Development Guide (3rd ed.)*. Addison-Wesley, 1992.
- [475] I. H. Witten, A. Moffat y T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, 1994.
- [476] I. H. Witten y E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers, 1999.
- [477] Y. Yang y R. Miller. Association rules over interval data. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1997.
- [478] M. J. Zaki. Scalable algorithms for association mining. En *IEEE Transactions on Knowledge and Data Engineering*, volumen 12, páginas 372–390, May/June 2000.
- [479] M. J. Zaki y C.-T. Ho, editors. *Large-Scale Parallel Data Mining*. Springer Verlag, 2000.
- [480] C. Zaniolo. Analysis and design of relational schemata. Informe técnico, Ph.D. Thesis, UCLA, TR UCLA-ENG-7669, 1976.
- [481] C. Zaniolo. Database relations with null values. *Journal of Computer and System Sciences*, 28(1):142–166, 1984.
- [482] C. Zaniolo. The database language GEM. En *Readings in Object-Oriented Databases*. eds. S.B. Zdonik and D. Maier, Morgan Kaufmann, 1990.
- [483] C. Zaniolo. Active database rules with transaction-conscious stable-model semantics. En *Intl. Conf. on Deductive and Object-Oriented Databases*, 1996.
- [484] C. Zaniolo, N. Arni y K. Ong. Negation and aggregates in recursive rules: the LDL++ approach. En *Intl. Conf. on Deductive and Object-Oriented Databases*, 1993.
- [485] C. Zaniolo, S. Ceri, C. Faloutsos, R. Snodgrass, V. Subrahmanian y R. Zicari. *Advanced Database Systems*. Morgan Kaufmann, 1997.
- [486] S. Zdonik, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul y D. Carney Monitoring streams—A new class of data management applications. En *Proc. Intl. Conf. on Very Large Data Bases*, 2002.
- [487] S. Zdonik y D. Maier (eds.). *Readings in Object-Oriented Databases*. Morgan Kaufmann, 1990.
- [488] T. Zhang, R. Ramakrishnan y M. Livny. BIRCH: an efficient data clustering method for very large databases. En *Proc. ACM SIGMOD Conf. on Management of Data*, 1996.
- [489] Y. Zhao, P. Deshpande, J. F. Naughton y A. Shukla. Simultaneous optimization and evaluation of multiple dimensional queries. En *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 1998.

- [490] Y. Zhuge, H. Garcia-Molina, J. Hammer y J. Widom. View maintenance in a warehousing environment. En *Proc. ACM SIGMOD Conf. on the Management of Data*, 1995.
- [491] M. M. Zloof. Query-by-example: a database language. *IBM Systems Journal*, 16(4):324–343, 1977.
- [492] J. Zobel, A. Moffat y K. Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23, 1998.
- [493] J. Zobel, A. Moffat y R. Sacks-Davis. An efficient indexing technique for full text databases. En *Proc. Intl. Conf. on Very Large Databases, Morgan Kaufman pubs. (San Francisco, CA) 18, Vancouver*, 1992.

ÍNDICE

- 1NF, 370
2NF, 373
3FN, 371, 378, 381
4FN, 388
5FN, 390
A priori (propiedad), 557
Añadir tablas con SQL, 88
Abandonar privilegio, 448
Abortar, 255–256, 267
en cascada, 262
transacción, 266
Abstracción, 521
Abstracción de página, 280
Acciones conflictivas, 259
Actualizable (vista), 85–86
Actualización de vistas
materializadas, 540
Actualización perdida, 261
Administrador de seguridad, 456
Administradores de bases de datos,
20
Advanced Encryption Standard
(AES), 457
AES, 457
Agregación en el modelo ER, 39, 82
Agregación en línea, 530
Agregación en SQL, 143, 155
Agregados definidos por el usuario,
497
Agrupación, 572
Agrupación en SQL, 146
Agrupación
jerárquica, 573
particional, 573
Agrupamiento, 282, 297
Aislamiento, 255
Ajuste, 28, 402, 404, 418
Ajuste de bases de datos, 21
Ajuste
de índices, 418
de bases de datos, 418
para la concurrencia, 427
Algoritmos no bloqueantes, 530
Almacenamiento de métodos en
memoria intermedia, 498
Almacenamiento de TAD y tipos
estructurados, 496
Almacenes de datos, 6, 428, 515, 535
actualizar, 536
cargar, 536
extraer, 535
limpiar, 536
metadatos, 536
purgar, 536
transformar, 536
ALTER, 445
Alternativas de entradas de datos en
un índice, 282
Altura de un árbol, 287, 314
Ampliación de los optimizadores, 500
Análisis explorador de datos,
515–516, 554
Anidamiento (operación), 482
ANSI, 6, 58
API (Application Programming
Interface), 181
Aprendizaje de la máquina, 554
Arboles
árbol B+, 313
clasificación y regresión, 568
de decisión, 568
división de atributos, 569
poda, 569
ISAM, 310
regresión, 568
Archivo de publicaciones, 595
Archivos, 19
Archivos de firmas, 597
de cortes de bits, 598
Archivos
agrupados, 283, 291
asociativos, 284
de montículo, 281, 289
de registros, 281
en montículos, 19
ordenados, 290
Armstrong (axiomas), 367
Arquitectura de un SGBD, 18
Arquitecturas de aplicaciones, 220
cliente-servidor, 221
de dos capas, 221
de tres capas, 222
capa de presentación, 223
capa intermedia, 224
de una capa, 220
Arrays, 479
fragmentos, 496, 534
ASE de Sybase, 456, 475
Asertos en SQL, 158
Asesores sobre índices, 414
Asistentes
ajuste de índices, 414
para ajuste, 416
Asociación (reglas), 561
Asociación
dinámica, 342, 347
estática, 340
extensible, 342
duplicación de directorio, 344
profundidad global, 344
profundidad local, 345
lineal, 347
contador de nivel, 348
familia de funciones de asociación,
347
Atómica (fórmula), 113
Atomicidad, 254–255
Atributo, 10
categórico, 567
dependiente, 567
numérico, 567
predictor, 567
Atributos en el modelo ER, 28
Atributos en el modelo relacional, 59
Atributos en XML, 213
Autenticación, 443
Autocompromiso en JDBC, 184
Autoridades, 600
Autoridades de certificación, 459
Autorización, 9, 20, 453
basada en roles, 445
AVG, 144
Axiomas
completos, 368
para DF, 367
seguros, 368
Ayuda a la toma de decisiones,
513–514
Búsqueda en lenguaje natural, 591
B+ (árboles), 287, 313
altura, 287, 314
búsqueda, 315
borrado, 320
carga masiva, 328
compresión de claves, 326
conjunto secuencia, 313
formato de un nodo, 315
frente a ISAM, 296
inserción, 317
orden, 314
B2F estricto, 263
Base (tabla), 84
Base de datos relacional

- ejemplar, 61
- esquema, 61
- Bases de datos, 4
- Bases de datos activas, 127, 159
- Bases de datos estadísticas, 522
- Bases de datos
 - ajuste, 404
- Bases de datos
 - diseño
 - para un SGBDROO, 491
 - refinamiento de esquemas, 361
 - estadísticas, 461
 - Internet, 7
- Bell-LaPadula (modelo de seguridad), 453
- Biblioteca DBI, 234
- BIRCH, 574
- BLOB, 474, 496
- Bloqueo, 266
 - B2F estricto, 263
 - bloqueos compartidos, 263
 - bloqueos exclusivos, 263
- Bloqueos, 17, 530
 - compartidos, 263
 - conurrencia, 427
 - consecuencias sobre el rendimiento, 427
 - exclusivos, 263
- Bolsas, 478, 480
- Boyce-Codd (forma normal), 370, 376
- Cálculo relacional, 112
 - de dominios, 117
 - de tuplas, 112
 - potencia expresiva, 119
 - seguridad, 119
- Cadenas de caracteres en SQL, 133
- Cajones, 284
 - en un archivo asociativo, 340
- Calendarios, 563
- Calientes (puntos), 424, 427, 429
- Campo, 59
- Canales ocultos, 455
- Candidata (clave), 64, 74
- Capa de Sockets Seguros, 208
- Cardinalidad de una relación, 61
- Carga de trabajo, 296
- Carga de trabajo y diseño de bases de datos, 402
- Carga masiva de árboles B+, 328
- Carro de la compra, 556
- CASCADE en claves externas, 70
- CGI, 233
- Cierre
 - de atributos, 369
 - de DF, 367
- Cifrado, 456, 458
 - de clave pública, 457
 - RSA, 457
 - simétrico, 457
- Clases
 - interfaz, 502
- Clasificación (árboles), 568
- Clasificación, 567
- Clave (restricción), 64–65
- Clave de sesión, 459
- Claves, 29, 366
 - externas
 - eidos, 493
- Claves
 - candidata, 64, 74
 - candidatas, 29
 - candidatas frente a búsqueda, 285
 - de búsqueda, 281
 - compuestas, 299–300
 - concatenadas, 299–300
 - externa, 66, 74
 - primarias, 29, 65
- Clientes ligeros, 221
- CLOB, 475
- CODASYL, D.B.T.G., 630
- Cola de conexiones, 186
- Colecciones anidadas, 481, 494
- Colisiones, 347
- Columna, 59
- Comercio electrónico, 205
- Comparación de conjuntos en SQL, 141
 - Compatible para la unión, 101
 - Compleitud relacional, 120
 - Compresión de claves, 326
 - Compresión en árboles B+, 326
 - Comprometer, 267
 - Comunicaciones entre procesos (IPC), 498
 - Concesión de privilegios en SQL, 448
 - Concreción, 521
 - Concurrencia, 9, 16
 - Conexiones en JDBC, 184
 - Configuración de índices, 414
 - Conjunto de entidades débiles, 36
 - Conjunto de relaciones ejemplar, 30
 - Conjunto secuencia en un árbol B+, 313
 - Conjuntos (operaciones)
 - en el álgebra relacional, 101
 - Conjuntos diferencia, 101
 - Conservación de dependencias en la descomposición, 375
 - Consistencia, 254–255
 - Constructores de tipos, 478
 - Consultas, 15
 - anidadas, 138
 - correlacionadas, 140
 - ejemplo
 - C1, 107, 115, 118, 131, 138, 140, 146
 - C10, 112
 - C11, 113, 118, 129
 - C12, 115
 - C13, 115
 - Consultas de ejemplo
 - C1, 107, 115, 118, 131, 138, 140, 146
 - C10, 112
 - C11, 113, 118, 129
 - C12, 115
 - C13, 115
 - C14, 116, 119
 - C15, 129
 - C16, 132
 - C17, 134
 - C18, 134
 - C19, 137
 - C2, 108, 116, 118, 133, 139
 - C20, 137
 - C21, 139
 - C22, 141
 - C23, 141
 - C24, 141
 - C25, 144
 - C26, 144
 - C27, 144
 - C28, 145
 - C29, 145
 - C3, 109, 133
 - C30, 145
 - C31, 146
 - C32, 147
 - C33, 150
 - C34, 150
 - C35, 151
 - C36, 152
 - C37, 152
 - C4, 109, 133
 - C5, 109, 135
 - C6, 110, 135, 142
 - C7, 111, 116, 118
 - C8, 111
 - C9, 111, 116, 119, 143
- Consultas ventana, 525
- Consultas
 - ajuste, 420
 - booleanas, 590
 - de clasificación, 590
 - iceberg, 559
 - por rangos, 299
 - seguras, 119
- Contador de nivel en asociación lineal, 348
- Contenido XML, 216
- Control de acceso, 9, 442–443
 - discrecional, 443
 - obligatorio, 444
 - objetos y sujetos, 453
- Controlador, 181–182
 - gestor, 182
 - tipos, 182
- Cookies, 235, 241
- Copos de nieve (consultas), 534
- Corrientes de datos, 577
- COUNT, 144
- Creación de relaciones en SQL, 61
- CREATE (orden)
 - SQL, 445
- CREATE DOMAIN, 157
- CREATE TABLE, 62
- CREATE TRIGGER, 160
- CREATE TYPE, 157

- CREATE VIEW, 84
 CS564 en Wisconsin, xxix
 CSS, 231
 Cuantificadores, 113
 Cuarta forma normal, 388
 CUBE (operador), 523, 534, 550
 Cumplimiento de restricciones de integridad, 69
 Cursos
 actualizables, 179
 en SQL, 177, 179
 CVA (conjunto), 571
 Data Encryption Standard, 457
 Datos semiestructurados, 605
 DB2 de IBM, 158, 456, 475, 479, 487, 534, 545
 DB2
 Asesor de índices, 416
 DBA, 20
 Decisión (árboles), 568–569
 división de atributos, 569
 poda, 569
 Declaraciones de tipo de documento (DTD), 215
 Deducción y seguridad, 462
 Dependencias
 de inclusión, 390
 de reunión, 390
 funcionales, 366
 axiomas de Armstrong, 367
 cierre, 367
 cierre de atributos, 369
 proyección, 375
 recubrimiento mínimo, 378
 multivaloradas, 386
 parciales, 372
 transitivas, 372
 Dependiente (atributo), 567
 DES (Data Encryption Standard), 457
 Desanidamiento (operación), 481
 Descifrado, 456
 Descomposiciones, 364
 conservación de dependencias, 375
 en 3FN, 378
 en FNBC, 376
 horizontal, 424
 sin pérdida, 373
 sin redundancia, 424
 vertical, 420, 424
 Descriptor de privilegio, 449
 Descubrimiento del conocimiento, 554
 Desnormalización, 404, 420, 422
 Detección de cambios, 578
 Diccionario, 595
 Diferencia (operación), 135
 Diferencia de conjuntos (operación), 101
 Dimensiones, 516
 Discos
 tiempos de acceso, 289
 Diseño conceptual, 26
 ajuste, 419
 Diseño de bases de datos
 ajuste, 21, 28, 402, 418, 420
 análisis de requisitos, 26
 carga de trabajo, 402
 conceptual, 13
 consecuencias de la concurrencia, 427
 dependencias de inclusión, 390
 diseño conceptual, 26
 diseño físico, 28
 físico, 13, 296, 402
 formas normales, 369
 herramientas, 27
 para OLAP, 519
 refinamiento de los esquemas, 27
 valores null, 364
 Diseño físico de bases de datos, 28
 Diseño físico
 índices agrupados, 297
 índices de varios atributos, 300
 Diseño físico
 ajuste
 de índices, 418
 de consultas, 420, 425
 del esquema conceptual, 419
 asistente para ajuste, 414, 416
 carga de trabajo, 402
 consultas anidadas, 427
 decisiones, 404
 planes de índices, 413
 reducción de los puntos calientes, 428
 selección de índices, 405
 Disparadores, 127, 159
 activación, 159
 por filas, 160
 por filas e instrucciones, 160
 por instrucciones, 160
 Dispersas (columnas), 532
 División (operador), 105
 División de atributos, 569
 División
 en SQL, 143
 DM, 386
 Documentos XML
 bien formados, 215
 válidos, 215
 DoD (niveles de seguridad), 455
 Dominio (restricciones), 72
 Dominio, 29, 59
 DR, 390
 DROP, 445
 DTD, 215
 Duplicación de directorio, 344
 Duplicados en SQL, 131
 Duplicados en un índice, 284
 Durabilidad, 255
 Ejecución concurrente, 257
 Ejemplar de una relación, 59
 Elementos en XML, 213
 Eliminación de tablas con SQL, 88
 Encapsulación, 483
 Entidades, 4, 12, 28
 Entidades débiles, 35
 Entidades
 débiles, 80
 Entradas de índice, 309
 Entradas de datos en un índice, 282
 Equirreunión, 104
 ERP, 7
 ES (jerarquía), 562
 Escalabilidad, 554
 Escritura ciega, 261
 Especialización, 37
 Esquema, 10, 59, 61
 Esquema en estrella, 519
 Esquema XML, 218
 Esquema
 ajuste, 419
 conceptual, 12
 evolución, 419
 externo, 13
 físico, 13
 lógico, 12, 27
 Esquema
 refinamiento
 desnormalización, 422
 Estrategia de evaluación conceptual, 128
 Etiquetas en HTML, 210
 Evaluación de índice, 297
 Eventos que activan disparadores, 159
 Evitar abortos en cascada, 262
 Evolución de bloques de datos, 577
 EXEC SQL, 175
 Expresiones de ruta, 479, 607
 Expresiones en SQL, 133, 154
 Extensibilidad
 indexación de nuevos tipos, 496
 Externa (clave), 66, 74
 Externas (reuniones), 155
 Fórmulas, 113
 Factor de reducción, 499
 Falsos positivos, 598
 Fantasmas, 270
 SQL, 270
 Fecha y hora en SQL, 134
 Firmas digitales, 460
 FNBC, 370, 376
 Formas normales, 369
 1NF, 370
 2NF, 373
 Formas normales
 3FN
 síntesis, 381
 3NF, 371
 4FN, 388
 5FN, 390
 ajuste, 419

- dominio-clave, 399
- FNBC, 370
- FNPR, 399
- normalización, 376
 - proyección-reunión, 399
- Formularios HTML, 225
- Fragmentos, 496
- Fragmentos de arrays, 534
- Frecuencia del término, 591
- Frecuencia inversa en el documento, 592
- Frecuente (lote), 557
- Frontera negativa, 580
- Fuente de datos, 182
- Función distancia, 572
- Funciones de agregación
 - en SGBDROO, 497
- Funciones de asociación, 284, 341, 347
- Funciones de un solo sentido, 457
- Generalización, 38
- Gestión de sesiones, 235
- Gestor
 - de bloqueos, 19
 - de la memoria intermedia, 19
 - de recuperaciones, 19
 - de transacciones, 19
 - del espacio de disco, 19
- GiST, 497
- Grado de salida, 287, 314, 326, 328
- Grafo de autorizaciones, 449
- GRANT (orden)
 - SQL, 444, 448
- GRANT OPTION, 444
- Guías de datos, 616
- Herencia de atributos, 37
- Herencia en bases de datos de objetos, 485
- Hojas de estilo, 231
 - en cascada, 231
- Horizontal (descomposición), 424
- HTML, 210, 212
 - etiquetas, 210
 - formularios, 225
- HTTP
 - ausencia de estado, 238
 - respuesta, 209
 - solicitud, 209
- IBM DB2, 325, 327
- Iceberg (consultas), 559
- Identificador de registro, 281
- Identificador uniforme de recursos (URI), 206
- Identificadores de autorizaciones, 445
- Identificadores de objetos, 487
- Idos, 487
 - integridad referencial, 493
 - y claves externas, 493
 - y URL, 489
- Idr, 281
- IDS, 6
- Igualdad
 - en profundidad y superficialmente, 488
- IMS, 6
- Independencia con respecto a los datos, 8, 14
 - física, 14
 - lógica, 14
 - vistas, 14
- Independencia de datos lógica, 85
- Indexación, 13
- Indexación de nuevos tipos de datos, 496
- Indexación
 - clave concatenada, 299
 - entradas de datos duplicadas, 284
 - índice invertido, 595
 - índices, 281
 - únicos, 284
 - agrupado vs. no agrupado, 282
 - asociativos, 284
 - no agrupados, 292
 - primario vs. secundario, 283
 - primarios, 283
 - secundarios, 283
 - árbol, 285
 - árbol B+, 313
 - alternativas de entradas de datos, 282
- Indexación
 - asociación
 - estática, 340
 - extensible, 342
 - lineal, 347
 - asociativo, 340
 - cajones, 340
 - funciones de asociación, 341
 - páginas primarias y de desbordamiento, 340
 - basada en árboles, 285
 - clave compuesta, 299
- Indexación
 - consulta
 - de igualdad, 299
- Indexación
 - consultas
 - por rangos e índices de clave compuesta, 299
 - correspondencia de una selección, 300
 - dinámica, 313, 342, 347
 - en SQL, 302
 - entrada de datos, 282
 - estática, 310
 - grado de salida, 287
 - ISAM, 310
 - mapas de bits, 531
 - de reunión, 534
 - no agrupado, 293
- selección de igualdad frente a rango, 296
- Informix, 327, 456, 475, 479, 531, 534
- Instantáneas, 545
- Integridad referencial, 69
 - en SQL, 69
 - idos, 493
 - opciones de violación, 69
- Intelligent Miner, 575
- Interbloqueo, 266
- Interfaz de clase, 502
- Interfaz de programación de aplicaciones, 181
- Interfaz servlet, 236
- Intersección (operación), 101, 135
- IQ de Sybase, 531, 534
- ISAM, 296, 310
- ISO, 6, 58
- Java Database Connectivity (JDBC), 181, 204
- Java
 - máquina virtual, 483
 - servlet, 236
- JavaScript, 228
- JDBC, 181, 184, 204, 535
 - arquitectura, 182
 - autocompromiso, 184
 - avisos, 189
 - clase DatabaseMetaData, 190
 - clase PreparedStatement, 186
 - clase ResultSet, 187
 - conexión, 184
 - excepciones, 189
 - fuente de datos, 182
 - gestión de controladores, 184
 - gestor de controladores, 182
- Jerarquías de colecciones, 487
- Jerarquías
 - de clases, 37, 81
 - de herencia, 37, 81
- ES, 37
- KDD, 555
- LDD, 12
- Lectura
 - desfasada, 260
 - no repetible, 261
- Legal (ejemplar de relación), 63
- Lenguaje anfitrión, 174
- Lenguaje de consulta, 72, 97
 - álgebra relacional, 98
 - cálculo relacional de dominios, 117
 - cálculo relacional de tuplas, 112
 - completitud relacional, 120
 - OQL, 503
 - SQL, 125
 - XQuery, 607
- Lenguaje de definición de datos (LDD), 61, 126
- Lenguaje de estilo extensible (XSL), 212

- Lenguaje de manipulación de datos (LMD), 126
- Lenguaje de manipulación de objetos, 502
- Lenguaje de marcado, 210
de hipertexto (HTML), 210, 212
extensible (XML), 212–213, 215–216
generalizado estándar (SGML), 212
- Lenguaje
anfitrión, 15
de consulta, 15
de definición de datos, 12
de manipulación de datos, 15
- LMD, 15
- LOB, 475
- Localizador, 475
- Localizador universal de recursos (URL), 208
- Lotes, 557
frecuentes, 557
a priori (propiedad), 557
soporte, 557
- Métodos
almacenamiento en memoria intermedia, 498
interpretados y compilados, 498
seguridad, 498
- Módulos Perl, 234
- Manejador de eventos, 230
- Mantenimiento de modelos, 578
- Mapas de bits (índices), 531
- Materialización de vistas, 538
- MathML, 218
- MAX, 144
- Medidas, 516
- Metadatos, 536
- Microsoft SQL Server, 325, 327, 475
- MIN, 144
- Minería de datos, 516, 553
- Modelo de datos, 10
en red, 6
jerárquico, 6
multidimensional, 516
relacional, 6, 10
semántico, 10, 27
- Modelo de espacio vectorial, 591
- Modelo ER
agregación, 39, 82
atributos, 28
atributos descriptivos, 30
claves, 29
dominios de los atributos, 29
entidades débiles, 35, 80
entidades y conjuntos de entidades, 28
jerarquías de clases, 37, 81
papeles, 32
relaciones, 29
una a varias, 33
varias a una, 33
- varias a varias, 33
- restricción de clave, 32
- restricciones de participación, 34, 78
- solapamiento y cobertura, 38
- Modelo relacional, 57
- Modificación de consultas, 538
- Modificación de tablas con SQL, 62
- Modo de acceso en SQL, 270
- MOLAP, 517
- MRP, 7
- Multiconjuntos, 129, 478, 480
- Natural (reunión), 105
- Nivel de aislamiento, 185
- Nivel de aislamiento en SQL, 270
- READ COMMITTED, 270
- READ UNCOMMITTED, 270
- REPEATABLE READ, 270
- SERIALIZABLE, 270
- Niveles de abstracción, 12
- Niveles de seguridad, 456
- NO ACTION en claves externas, 70
- Nodos, 600
- Nombres de restricciones en SQL, 65
- Normalización, 58, 376, 404
- Normalización mediante cosenos, 592
- Null (valores), 364
- Null
en SQL, 67–68, 70
- Nulos
en SQL, 153
- Objeto de gran tamaño, 475
de caracteres, 475
- ODBC, 181, 204
- ODL, 501
- ODMG (modelo de datos)
atributos, 501
clases, 501
métodos, 501
objetos, 501
relación inversa, 501
relaciones, 501
- OEM, 606
- OLAP, 433, 515–516, 550
abstracción y concreción, 521
consultas ventana de SQL, 525
diseño de bases de datos, 519
pivotaje, 521
tabla de doble entrada, 521
tabla de hechos, 517
tablas de dimensiones, 519
- OLTP, 514
- OML, 502
- Open Database Connectivity (ODBC), 181, 204
- Operadores de conjuntos en SQL, 135
- Optimizador de consultas, 19
- Optimizadores
ampliación, 500
para SGBDROO, 499
predicados costosos, 500
- OQL, 501, 503
- Oracle, 27, 325, 327, 456, 475, 479, 487, 500, 531, 534, 545
- Orden de un árbol B+, 314
- Ordenación en SQL, 134
- Organización de archivo, 280
árbol, 285
agrupado, 291
aleatoria, 289
asociativa, 284
indexado, 282
ordenado, 290
- Orientados a objetos (SGBD), 472, 501, 504
- Página primaria de un cajón, 284
- Página primaria frente a página de desbordamiento, 340
- Palabras de parada, 592
- Papeles en el modelo ER, 32
- Paquetes en SQL:1999, 126
- Particiones
verticales, 404
- Participación (restricciones), 78
- Participación
parcial, 34
total, 34
- Patrones secuenciales, 564
- Pivoteaje, 521
- Plan de ejecución, 19
- Planes de índices, 413
- Planificaciones, 257
ACA, 262
evitar abortos en cascada, 262
recuperables, 262
secuenciales, 258, 262
secuenciales, 257
- PMML, 555
- Poda, 569
- Poliinstanciación, 455
- Potencia expresiva
álgebra y cálculo, 119
- Precisión, 594
- Precoz (vinculación), 486
- Predicados costosos, 500
- Predictor (atributo), 567
- Premio Turing, 6
- Primarias (claves), 65
en SQL, 65
- PRIMARY KEY (restricción de SQL), 65
- Primera forma normal, 370
- Principio de sustitución, 486
- Privilegios de acceso, 443
- Procedimientos almacenados, 194
- Procesamiento analítico en línea (OLAP), 515
- Procesamiento de transacciones en Línea (OLTP), 514
- Procesamiento del lado del servidor, 236

- Proceso del descubrimiento de conocimiento, 555
- Productividad, 258
- Producto aspa (operación), 101
- Producto cartesiano, 101
- Profundidad (igualdad), 488
- Profundidad global en la asociación extensible, 344
- Profundidad local en la asociación extensible, 345
- Programadores de aplicaciones, 20
- Propietaria identificadora de una entidad débil, 36
- Protocolo de bloqueo, 263
- Protocolos de bloqueo, 17
- Protocolos de comunicación, 208
 - HTTP, 208
 - sin estado, 210
- Proyecciones
 - definición, 100
- Pruebas de rendimiento, 432, 440
- Pugna, 266
- Puntos calientes, 424, 427, 429
- Puntos de almacenamiento, 268
- Puntos de revisión, 17
- Quinta forma normal, 390
- Réplica
 - asíncrona, 536
- Raíz de un documento XML, 215
- Rastreador Web, 599
- Recubrimiento mínimo, 378
- Recuperabilidad, 262
- Recuperación, 9, 21, 594
- Recuperación de información, 587
- Recuperación
 - en caso de fallo, 9, 17, 21
 - registro, 17
 - registro histórico, 256
- Redundancia y anomalías, 363
- Referencia (tipos), 492
- Referencias de entidad en XML, 213
- Refinamiento de esquemas, 361
- Refinamiento de los esquemas, 27
- Registro, 17
- Registro histórico, 256
- Registro
 - de escritura previa, 17
 - WAL, 17
- Registros, 59
- Registros de datos, 10
- Reglas
 - de asociación, 561, 563
 - con calendarios, 563
 - con jerarquías, 562
 - para la predicción, 565
 - de clasificación, 567
 - de regresión, 567
- Regresión (árboles), 568
- Relación, 59
 - cardinalidad, 61
 - ejemplar, 59
- ejemplar legal, 63
- esquema, 59
- grado, 61
- Relacional (álgebra), 98–99
 - división, 105
 - expresión, 99
 - operaciones con conjuntos, 101
 - potencia expresiva, 119
 - proyección, 100
 - renombramiento, 103
 - reunión, 103
 - selección, 99
- Relacional orientado a objetos (SGBD), 504
- Relacionales y orientados a objetos (SGBD), 472
- Relaciones, 4, 10, 12, 29, 33
- Relaciones anidadas
 - anidamiento, 482
 - desanidamiento, 481
- Relaciones multinivel, 454
- Renombramiento en el álgebra relacional, 103
- Rescate de punteros, 499
- Restricciones de dominio, 157
- Restricciones de integridad, 9, 11, 32, 34, 38, 63, 78
 - clave externa, 66
 - claves, 64
 - de clave, 32
 - de cobertura, 38
 - de dominio, 29
 - de participación, 34
 - de solapamiento, 38
 - dominio, 60, 72
 - en SQL, 156–158
 - traducción a SQL, 71
- Reunión en estrella (consultas), 534
- Reuniones, 103
 - definición, 103
 - equirreunión, 104
 - externas, 155
 - reunión natural, 105
- REVOKE (orden)
 - SQL, 447–448
- ROLAP, 519
- ROLLUP, 524
- SABRE, 6
- Secuencia
 - de consulta, 576
 - de lotes, 564
- Secuencialidad, 258, 262
- Secure Electronic Transaction, 460
- Secure Sockets Layer, 458
- Segunda forma normal, 373
- Seguridad, 20, 443, 445
 - autenticación, 443
 - bases de datos estadísticas, 461
 - cifrado, 458
 - clases, 444, 453
- Seguridad
 - control de acceso
 - discrecional, 443
 - control obligatorio de acceso, 444
 - de los métodos, 498
 - deducción, 462
 - mecanismo, 442
 - política, 442
 - privilegios, 443
 - uso de vistas, 451
- Selección de índices, 405
- Selección de división, 570
- Selección
 - de igualdad, 296
 - por rango, 296
- Selecciones
 - definición, 99
- Servicios Web, 207
- Servidores de aplicaciones, 233, 235
- Servlet, 236
 - request, 237
 - response, 237
- SET (protocolo), 460
- SET DEFAULT en claves externas, 70
- SGBD, 4
 - arquitectura, 18
- SGBDOO y SGBDROO, 505
- SGBDR y SGBDROO, 505
- SGBDROO
 - diseño de bases de datos, 491
 - implementación, 495
 - y SGBDOO, 505
 - y SGBDR, 505
- SGML, 212
- Sin pérdida (descomposición), 373
- Sistema gestor de bases de datos, 4
- Sobrecarga, 486
- Soporte, 557
 - clasificación y regresión, 567
 - lote frecuente, 557
 - reglas de asociación, 561
 - secuencia de lotes, 564
- SQL dinámico, 181
- SQL incorporado, 174
- SQL Server de Microsoft, 416, 456, 531, 534, 545
- SQL Server
 - minería de datos, 575
- SQL
 - actualizaciones de vistas, 85
 - ALL, 141, 146
 - ALTER, 445
 - ALTER TABLE, 89
 - ANY, 141, 146
 - AS, 133
 - AVG, 144
 - BETWEEN, 409
 - cadenas de caracteres, 133
 - CARDINALITY, 480
 - CASCADE, 70
 - COMMIT, 267

- consultas correlacionadas, 140
COUNT, 144
 creación de vistas, 84
CREATE, 445
CREATE DOMAIN, 157
CREATE TABLE, 62
CUBE, 523
 cursores, 177
 actualizables, 179
 mantenibles, 179
 ordenación, 180
 sensibilidad, 179
DATE, 134
DELETE, 68
 dinámico, 181
DISTINCT, 129, 131
DISTINCT en la agregación, 144
 división, 143
DROP, 445
DROP TABLE, 89
EXCEPT, 135, 142
EXEC, 175
EXISTS, 135, 155
 expresiones, 133, 154
 fantasmas, 270
GRANT, 444, 448
 `GRANT OPTION`, 444
GROUP BY, 146
HAVING, 146
 identificadores de autorizaciones, 445
IN, 135
 indexación, 302
INSERT, 62, 68
SQL
 integridad referencial
 cumplimiento, 69
INTERSECT, 135, 142
IS NULL, 154
 lenguaje de definición de datos (LDD), 61, 126
 lenguaje de manipulación de datos (LMD), 126
MAX, 144
MIN, 144
 modo de acceso, 270
 multiconjuntos, 129
 nivel de aislamiento, 270
NO ACTION, 70
 nombres de restricciones, 65
 normalización, 58
 normas, 170
NOT, 130
 nulos, 153
 operaciones de agregación, 155
 definición, 143
 ordenación, 134
ORDER BY, 180
 paquetes, 126
 privilegios, 443–444
 `DELETE`, 444
 `INSERT`, 444
 `REFERENCES`, 444
 `SELECT`, 444
 `UPDATE`, 444
 programación con lenguajes incorporados, 175
 puntos de almacenamiento, 268
READ UNCOMMITTED, 270
REPEATABLE READ, 270
SQL
 restricciones de integridad
 assertos, 68, 158
 `CHECK`, 156
 comprobación diferida, 71
 efecto sobre las modificaciones, 68
 `PRIMARY KEY`, 65
 restricciones de dominio, 157
 restricciones de tabla, 68, 156
 `UNIQUE`, 65
 reuniones externas, 155
 `REVOKE`, 447–448
 `CASCADE`, 448
 `ROLLBACK`, 267
 `ROLLUP`, 524
 seguridad, 445
 `SELECT-FROM-WHERE`, 129
 `SERIALIZABLE`, 270
 `SOME`, 141
 soporte de transacciones, 267
 `SQLCODE`, 178
 `SQLERROR`, 176
 `SQLSTATE`, 176
 SQL
 subconsultas anidadas
 definición, 138
 `SUM`, 144
 tipos distintos, 157
 transacciones en cadena, 268
 transacciones y restricciones, 71
 `UNION`, 135
 `UNIQUE`, 155
 `UPDATE`, 62, 69
 valores nulos, 67–68, 70
 vistas, 87
 actualizables, 86–87
 insertables, 87
SQL/MM
 Data Mining, 555
 entorno, 474
 Full Text, 603
SQL/PSM, 197
SQL/XML, 607
SQL:1999, 58, 170, 501, 511
 autorización basada en roles, 445
SQL:1999
 constructores de tipos
 array, 478
 fila, 478
 disparadores, 159
 tipos estructurados, 479
 definidos por el usuario, 478
 tipos referencia e idos, 487
SQL:2003, 170
SQLCODE, 178
SQLERROR, 176
SQLJ, 190
 iteradores, 193
SQLSTATE, 176
SRQL, 550
SSL (protocolo), 458
SSL, 208
Subclase, 37
 Sublenguaje de datos, 15
SUM, 144
Superclase, 37
Superclave, 64
Superclaves, 367
 Superficial (igualdad), 488
Sybase, 27
 Sybase ASE, 325, 327
System R, 6
 Término de búsqueda, 589
 Tabla de doble entrada, 521
 Tablas, 59
 base, 84
TAD, 482–483
 almacenamiento, 496
 encapsulación, 483
 Tardía (vinculación), 486
 Tercera forma normal, 371, 378, 381
 Tiempo de petición en discos, 289
 Tiempo de respuesta, 258
 Tiempos de acceso a disco, 289
 Tipos de contenido en XML, 216
 Tipos distintos en SQL, 157
Tipos
 abstractos de datos, 482–483
 colección, 479
 complejo y referencia, 492
 complejos, 477, 492
 de datos masivos, 479
 definidos por el usuario, 482
 estructurados, 479
 almacenamiento, 496
 definidos por el usuario, 478
 igualdad de objetos, 488
 opacos, 483
 referencia, 492
 en SQL:1999, 487
Transacción
 propiedades, 16
Transacciones, 253–254
 abortadas, 256
 acciones conflictivas, 259
 ACID, 255
 anidadas, 268
 bloqueos y rendimiento, 427
 cliente, 556
 comprometidas, 256
 en cadena, 268
 en SQL, 267
 escritura, 256

- escritura ciega, 261
- lectura, 256
- planificación, 257
- propiedades, 255
- puntos de almacenamiento, 268
- restricciones en SQL, 71
- y JDBC, 184
- Travelocity, 6
- Traza de auditoría, 461
- Trivial (DF), 368
- Tupla, 59
- UDDI, 207
- UDS de Informix, 158, 487
- UML, 47
- UML
 - diagramas
 - de bases de datos, 48
 - de clases, 48
 - de componentes, 48
- Unión (operación), 101, 135
- Unicode, 214
- Unified Modeling Language, 47
- UNIQUE (restricción de SQL), 65
- URI, 206
- URL, 208
 - e idos, 489
 - JDBC, 184
- Vector documental, 591
- Vertical (descomposición), 420, 424
- Vinculación
 - dinámica, 483
 - tardía y precoz, 486
- Vistas, 13, 84, 87, 404
 - Vistas materializadas
 - actualizar, 540
 - Vistas
 - actualizables, 86–87
 - actualizaciones, 85
 - GRANT, 451
 - insertables, 87
- mantenimiento, 540, 545
 - incremental, 541
- materialización, 538
- modificación de consultas, 538
- para la seguridad, 451
- particiones, 545
- REVOKE, 451
- WSDL, 207
- XML
 - DTD, 215
 - elementos, 213
 - raíz, 215
 - referencias de entidad, 213
- XPath, 233
- XQuery, 607
 - expresiones de ruta, 607
- XSL, 212, 232
- XSLT, 232
- Zona caliente, 267

Sistemas de gestión de bases de datos se ha convertido rápidamente en uno de los textos líderes en asignaturas de bases de datos por su énfasis práctico y el amplio tratamiento de los temas tratados. La tercera edición incorpora nuevo material sobre el desarrollo de aplicaciones de bases de datos, incluyendo aplicaciones de Internet. Su enfoque práctico introduce a los estudiantes en los nuevos estándares, incluyendo JDBC, XML y arquitecturas de aplicaciones en tres capas. Su nueva y flexible organización permite que los profesores enseñen una asignatura orientada a las aplicaciones. Los capítulos introductorios permiten seleccionar de forma sencilla los capítulos que se necesiten, mientras que los capítulos avanzados de cada parte pueden ser opcionales.

Esta nueva edición también se caracteriza por sus mejoras pedagógicas (por ejemplo, los objetivos de cada capítulo y las preguntas de repaso) así como por un tratamiento actualizado y extendido de la minería de datos, asistentes para el ajuste de las bases de datos, ayuda a la toma de decisiones, recuperación de información, seguridad en Internet, bases de datos de objetos y gestión de datos XML. Sus contenidos se han revisado y ampliado para reflejar la nueva norma SQL:1999, incluyendo extensiones que dan soporte a los datos multimedia, bases de datos relacionales orientadas a objetos, OLAP, consultas recursivas, datos espaciales y SQL-J. Su organización y contenidos actualizados de temas avanzados también hacen a este libro ideal para usarlo durante dos asignaturas.

Raghuram Krishnan es profesor de informática en la Universidad de Wisconsin, Madison, y fundador y CTO de QUIQ. Es socio de ACM.

Johannes Gehrke es profesor ayudante de informática en la Universidad de Cornell.

“La tercera edición de *Sistemas de gestión de bases de datos* continúa la excelente tradición de las ediciones anteriores. La mezcla de teoría y práctica, junto a interesantes ejemplos, hacen un libro atractivo y fácil de leer. SQL se explica eficazmente y sus características importantes se tratan sin entrar en los detalles complicados de SQL. Estoy particularmente impresionado por los muy bien elegidos ejercicios y cuestiones, así como por la extensa bibliografía. ¡Es una excelente obra para cualquier estudiante o profesional de bases de datos!” *Jim Melton, editor de ISO/IEC 9075 (SQL), Oracle Corporation.*